Integrated Simulation Environment for Investigation of Multi-Agent Systems in Smart Grids Applications

Manswet Banka
Institute of Power Transmission and High Voltage Technology
University of Stuttgart
Stuttgart, Germany
manswet.banka@ieh.uni-stuttgart.de

Krzysztof Rudion
Institute of Power Transmission and High Voltage Technology
University of Stuttgart
Stuttgart, Germany
rudion@ieh.uni-stuttgart.de

***Abstract*— This paper presents a framework for simulation environment for modelling and testing smart grids controlled by multi agent systems. The environment is based on the co-simulation approach and consists of MATLAB/Simulink, where the power grid part is modelled, Java Agent DEvelopment framework, where the multi-agent system is programmed, and the interface between these two based on TCP/IP communication. The interface is designed to be general and it can be used also for purposes other than power systems application. Structure, integration within both ends, and functioning of the interface is the main focus of this paper.**

Keywords— co-simulation, multi-agent systems, JADE, MATLAB/Simulink, Smart Grids

I. Introduction

Nowadays the power systems tend to be increasingly decentralized and, therefore, there is a growing amount of devices which should be monitored and controlled, possibly in real time, in order to ensure stable and efficient operation of the system. State of the art in designing and optimizing of such systems is by means of simulations. There are different tools used to model power system domain and one of it is MATLB/Simulink with its Specialized Power Systems library [1]. In the control domain the multi-agent systems (MAS) have been gaining attention as the appropriate way of automated control of decentralized systems. A well-known environment for developing such platforms is Java Agent DEvelopment framework (JADE) [2]. There are however very limited options available for co-simulations between power system model and MAS. This paper presents the developed framework for integrating MATLAB/Simulink with JADE within one simulation environment.

The interfacing of JADE with MATLAB/Simulink is not straight forward. Reference [3] describes a tool which was designed for the data exchange between JADE and Simulink but the project is not actively supported anymore and the integration into newer Simulink's versions is troublesome. There are approaches like [4, 5], where JADE is interfaced with MATLAB but not Simulink. The authors in [6] proposed similar concept to the presented in this paper of interfacing JADE with Simulink using TCP sockets, however their design differs, since they proposed the communication over multiple TCP client-server connections. In this paper the connection is centralized and realized by only one TCP client on the Simulink model's side and only one TCP server on the JADE's side. One of the requirements for the framework is that the TCP client and server should be flexible and accept different number of agents exchanging data with Simulink and different number of exchanged signals without the need of additional connections. Although the framework was primary developed for application in power systems area its general character allows implementations in different areas as well.

II. Overview of the Architecture

The proposed solution consists on interfacing two separated simulation environment each dedicated to different domain. One of them is MATLAB/Simulink with its Specialized Power Systems library where the power system domain is modeled. The other one is JADE with the representation of the upper level control realized by agents. The clue is the bidirectional exchange of data in a systematized manner throughout the duration of the co-simulation. Although Simulink provides components allowing the user to log and observe data during a simulation, the external access is not straight forward. One of the possible and effective solutions is implementing a communication component of a standard protocol [6,7]. In this case the TCP/IP protocol is chosen, where the client is on the Simulink's side and server on the JADE's side. In the presented architecture the client collects signals within Simulink model which can represent different measurements required for taking control actions and sends them to the server at regular intervals. The server receives signals from the client and distributes them among existing agents, Fig. 1.

From the platform point of view the TCP server agent is a standard agent and communicates with others using provided in JADE means of communication – sending the Agent Communication Language (ACL) messages [5]. The block TCP client exchanges the data with other modeled blocks using standard Simulink signals connections. Although there can be different amount of blocks and agents on both sides, and even agents communicating with Simulink but not

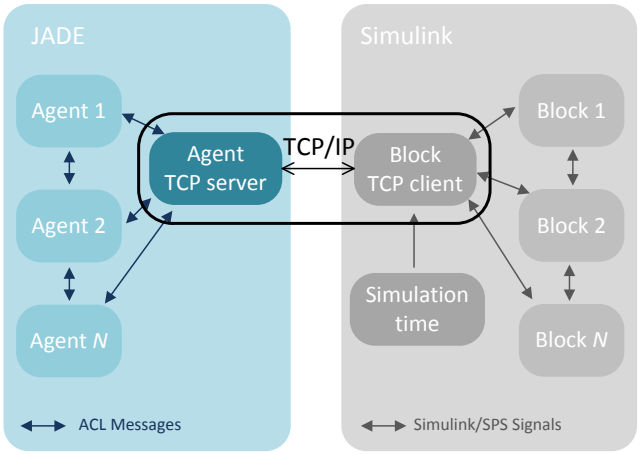

Fig. 1 Overview of the developed co-simulation architecture.

having its equivalent as a Simulink block, there is always only one server and one client who concentrate the data and manage their exchange. The crucial role plays "Simulation time" block on the Simulink side, Fig. 1, since it is the source of reference time for agents' operation and synchronization as described in section V.

III. TCP Client in Simulink

The TCP client is implemented on the Simulink side of the interface. Although there are ready-to-use blocks for sending and receiving over TCP/IP protocol provided in Simulink's libraries, the use of these in the proposed application requires special consideration and adjustments regarding time synchronization and processing of incoming and outgoing signals. Therefore, a customized C-programmed S-function block was used for this purpose [1]. The core functionality is owed to Windows's Winsocks, which provides C-language API and mechanisms for TCP communication. The main tasks of this block can be summarized:

- Establishing connection with the server,
- Collecting and sending time stamp to the server at regular time intervals specified by the user together with the current values of the Simulink signals,
- Receiving the co-simulation start flag and the set points from the agents,
- Providing the signals to the proper Simulink blocks.

Each Simulink block's code consists of numerous routines, which are called at different stages of the model execution. The most important ones for the implementation of the TCP client block are depicted in Fig. 2 together with the stages of the TCP connection.

The function *mdlInitializeConditions* is called only once to initialize the blocks parameters. In *mdlStart* the TCP socket is being opened and the connection to the server is established. This function is also called only once, at the beginning of the execution, which in turn means that the TCP connection remains open during the whole time of the co-simulation. The *mdlOutputs* function is called at every execution of the block. In its body the main part of data exchange takes place, Fig. 3. At the beginning of the simulation the block is awaiting the signal to start the co-simulation. Once the flag is received the Simulink time and signals are sent to the server. The block is now waiting for the response of the server and is blocking the execution of the rest of the model. Thanks to such a solution, which prevents Simulink engine from further calculations, there is no risk of performing them with outdated set points.

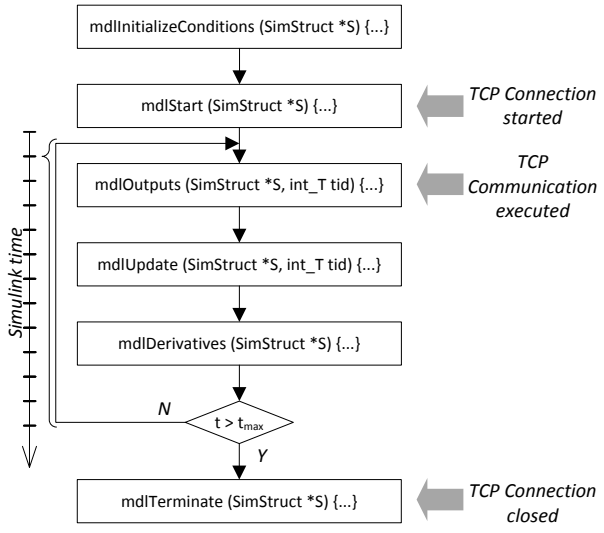

Fig. 2 The most important routines of a S-function in C-code.

The TCP client block, presented in Fig. 4, has two inputs and two outputs. The first input is foreseen for the clock providing the simulation time and is separated from other input signals. This signal is used by the client to determine the time stamp sent to the server. The second input takes all remaining signals, which should be sent to the agent platform. The ordering of these remaining signals has to be done manually by the user. On the other end of the interface the signals are alphabetically sorted according to the agents' names. However it is up to the user how the *n*-th signal of a specific agent will be interpreted by him. Therefore, the same ordering must be preserved in Simulink model. The first output is dedicated to the signal indicating the beginning of the co-simulation. Such a feature is relevant especially in case of bigger models with many state variables, when it is not always an easy task to determine the initial conditions for starting the model in a steady state. Therefore, when the user starts the Simulink model he can wait until the states converge to the steady state and only then start the co-simulation from the user interface in JADE. This event will be communicated to the TCP client block in Simulink and then propagated in the Simulink model as the mentioned flag signal. The block can be used with models of any size and complexity. Depending on these two factors and additionally the type of the solver and its step size the execution of the model might be very fast making this impossible for the user to react and start the co-simulation after starting the Simulink model. For this reason the TCP

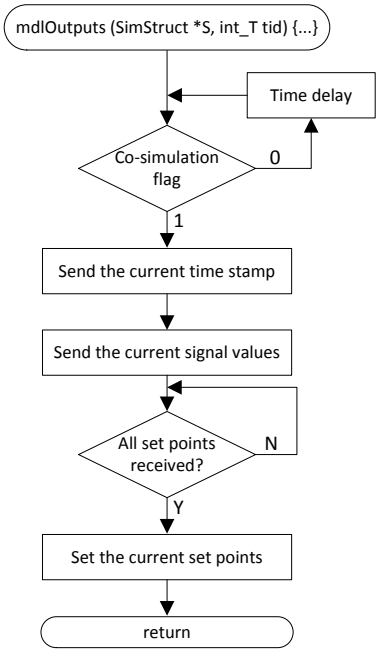

Fig. 3 Flow chart of the code responsible for the communication in the TCP/IP client block.

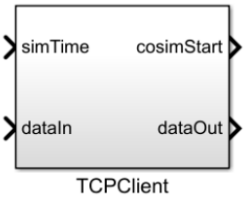

Fig. 4 Representation of the TCP client in Simulink model.

client block implements an artificial time delay during each execution, which can be easily adjusted by the user. For the more complex systems, whose execution is very slow anyway, the time delay can be set to 0 and for less complex and fast executed systems it can be increased to required number of seconds. As soon as the co-simulation was intentionally started by the user, the delay is omitted.

The Simulink's solver engine allows definition of time intervals at which particular blocks should be executed. By default it is each simulation step. However, it is not reasonable from the performance perspective and would slow down the co-simulation significantly, if the exchange of the data took place with resolution of milliseconds or less, depending on the modeled systems and its simulation step. Therefore, it is up to the user to specify the execution intervals by adjusting the "Sample time" parameter in the block's mask (t_{samp}). In the performed co-simulations value of 1s was used, which means that the communication took place and, therefore, the signals were updated every second of the Simulink time. Thanks to the Simulink scheduler it also applies to the variable step solvers.

IV. Agents and TCP Server in JADE

An inseparable feature of an agent that has to be realized in the presented framework is its ability of interacting with the environment. This can be other agents, data bases, human machine interface, sensors, etc. For the communication with other agents JADE provides a messaging system based on TCP/IP where ACL messages can be exchanged. This is a very convenient method of exchanging information between agents within the platform, since the implementation details are hidden from the programmer. In turn the TCP server is responsible for the sensors-like signals. It receives them from the client in Simulink model and then redistributes among agents using ACL messages. The server is implemented within a JADE agent, whose tasks can be summarized:

- Listening to the incoming client connection,
- Sending the co-simulation start flag to the client,
- Receiving the time stamp and current values of the Simulink signals. Redistributing them among the agents,
- Collecting the new set-points from the agents and sending to the client.

An important aspect to be considered is the way how the agents perform their tasks. In JADE the routines required to solve the tasks are defined in *Behaviour* classes. Among others there are:

- generic *Behaviour*
- *OneShotBehaviour*
- *CyclicBehaviour*

which differ from each other in their lifetime. *OneShotBehaviour* is executed only once, whereas the *CyclicBehaviour* is executed in a cyclic manner unless the programmer foresaw calling its killing routine. An object of the generic class *Behaviour* is destroyed automatically as soon as the tasks are completed, which may be done in several steps. In this framework another differentiation of the behaviour is proposed for the sake of time synchronization between Simulink and JADE as explained in section V:

- *Periodic behaviour* – performed at regular time intervals, e.g. measurement of a monitored parameter. It is implemented as generic *Behaviour* class.
- *Reactive behaviour* – performed only after request from other behaviour, within the same or different agent, e.g. providing specific data from a data base. It is implemented as *CyclicBehaviour* class.
- *CyclicTCPBehaviour* – used for communication between an agent and the TCP server agent. Implemented as *CyclicBehaviour* class.

The TCP server agent communicates with other agents via their *CyclicTCPBehaviour*. Hence, each agent communicating with Simulink in the proposed framework has to have an object of this class. This behaviour holds a list of all agent's *periodic behaviour* objects together with their execution period and the time of the next execution. Additionally, in order for an agent to be recognized by the server agent, it has to register the necessity of TCP communication at Directory Facilitator (DF) agent during creation [9].

For the server agent it is not important how often particular agents need the access to the signals and it updates all the values each time it receives them from the client, which means every t_{samp} specified by the user in the TCP client block.

As already mentioned in section III the signals exchange between the server and the client are sorted alphabetically according to the agents' names given by the user. Based on the number of signals for each type of agent predefined in a configuration file, data structures of appropriate size are created for signals incoming from Simulink and set points out going from JADE to Simulink model.

The server TCP agent is a typical JADE agent which means that it has a dedicated Java thread for itself but for the sake of parallel execution of the tasks at the MAS platform and listening to the request incoming from Simulink it runs a separated thread for this purpose.

V. Time Synchronization

Multithreading is a desired and inseparable feature of MAS. It is a challenge for co-simulation however. The time synchronization between agents' operation and the Simulink model execution is a main concern of the framework. As already signalized in section III depending on the chosen solver and size and complexity of the model the execution time may significantly vary. The agents developed in JADE run in real time according to the operating system's clock. The lack of common time source is emphasized especially when the agents need to perform their tasks at regular time intervals or need to analyze the time trends of measured signals. This issue is solved by using the Simulink time as

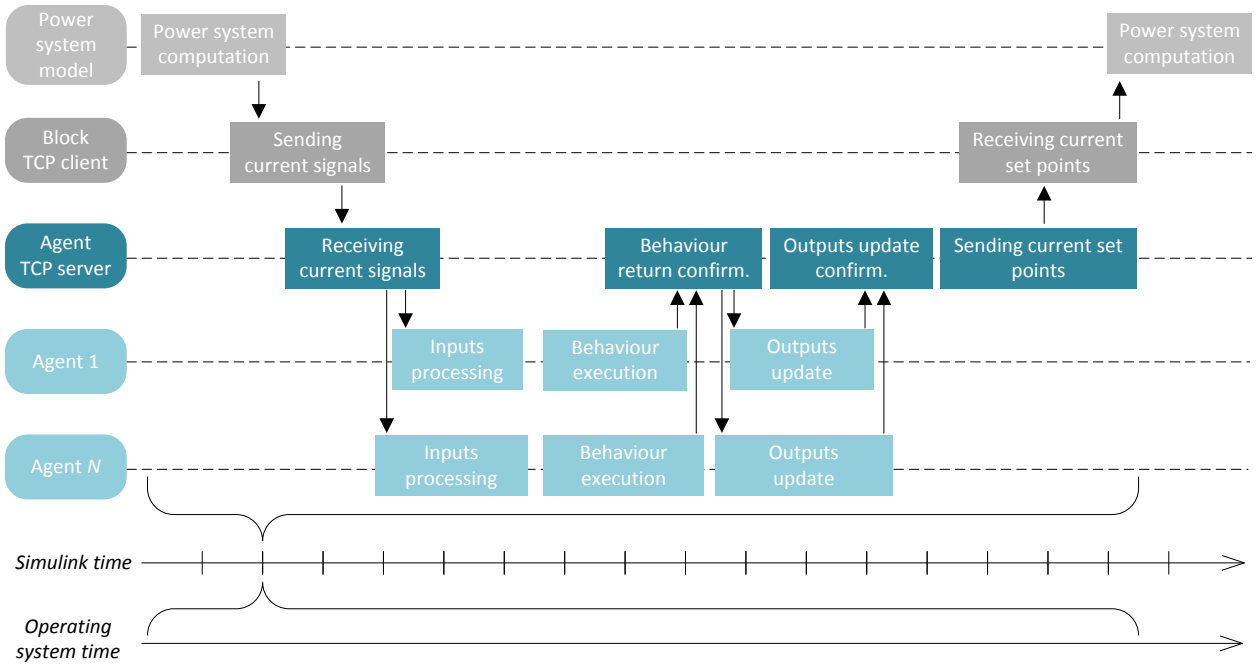

Fig. 5. Sequence of events during communication between JADE and Simulink.

the reference time and by the "return confirmation" between behaviour. Simulink time synchronization The Simulink model time is chosen to be the one to synchronize with. In the Fig. 5 one step of the co-simulation within presented framework is depicted in a schematic way. As per Fig. 2 the co-simulation was already started. When Simulink engine scheduler calls the TCP client block it sends over TCP/IP the time stamp and current values of Simulink signals to the server agent and is waiting for the server's response, blocking further computation in Simulink. The server redistributes the values among specific agents. Now the agents compare the current simulation time with their internal list of periodic behavior and decide whether or not to start them. If there is no scheduled behaviour for this period, the agent sends a message to the server agent in order to confirm that all its behaviour returned successfully. The confirmation is a vital part of the coordination from the server agent perspective. Without this it would be troublesome to ensure that no outputs would be updated after sending the values back to Simulink. In case an agent needs to perform some scheduled tasks all of them are required to send the confirmation back to the *CyclicTCPBehaviour* object after completing their actions, which is part of the "return confirmation" principle described in the next subsection. When all the considered agents confirmed successful return of their periodic behaviour, the TCP server agent requires them to submit the final values of the set points. Only after receiving responses from all the agents the server agent sends the current set points to the TCP client who should update its outputs with these values. The next simulation step in Simulink begins.

A. *Behaviour return confirmation*

All behaviour regardless if *periodic* or *reactive* must send a success confirmation to the calling behaviour, Fig. 6. This is an important difference in comparison to the native JADE framework, where behaviour normally completes without such a notification. In this framework it is called "return confirmation" principle and is based solely on ACL messages. The undeniable disadvantage of this principle is the fact that it limits the parallelization of the computation provided by a multithreaded environment, since each behaviour needs to wait for the end of its subbehaviour. Nonetheless, it helps ensuring that the results of all computations done by the agents would be sent to Simulink simultaneously and for the proper simulation instant, although the computations can differ in execution time greatly. An agent has to confirm the return only of all his own *periodic behaviour* to the server agent. The *reactive behaviour* is not considered here explicitly because they may be called on a request from another agent, *RB 1* of the *Agent N* is called by *PB n* of the *Agent 1* – Fig. 6. This means that an agent can send the confirmation to the server and at least one of its *reactive behaviour* may still not have returned because is dealing with providing some services to another agent. In the case like in Fig. 6 the *reactive behaviour* of the *Agent N* is implicitly considered as subbehaviour of a *periodic behaviour PB n* of an *Agent 1*. In general the behaviour dependencies can be summarized:

- *Periodic behaviour* is created within *TCPCyclicBehaviour* according to the time schedule. They have to confirm the successful end of operation.
- *Reactive behaviour (RB)* may be created at any moment and not only by agents communicating with the server. They have to confirm the successful end of operation to the requesting behaviour.
- *Periodic behaviour (PB)* does not communicate with other *periodic behaviour*. They may be self-sufficient and not need services from other behaviour, *PB n* of *Agent N*, or communicate with *reactive behaviour*

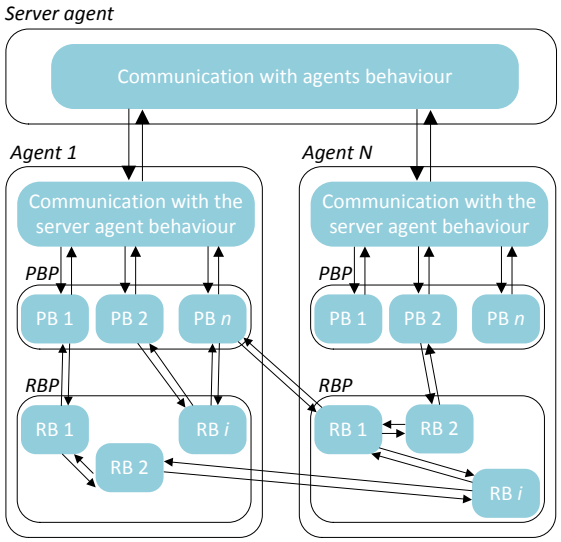

Fig. 6. Dependencies and communication between behaviour within the platform.

within one agent or not, *RB i* of *Agent 1* vs. *RB 1* of *Agent N*.

- *Periodic behaviour* may be designed to communicate only with other periodic behaviour, *RB 2* of *Agent 1*.
- *Reactive behaviour* can call subsequently other reactive behaviour but eventually only the return of the initial periodic behaviour must be confirmed to the server.

On the one hand the implementation of the return confirmation principle allows coordinating the co-simulation but on the other the system following such a principle differs from the reality since the measurements as well as set points are captured and set with exact the same time stamp (although some of the values might be used by the agents less frequently). However at the initial stages of developing MAS-based new control strategies and of proves of concept such a feature is not a vital issue.

B. Simulink's variable step solver

Simulink model can be computed using one of the implemented fixed or variable step solvers. It is not possible to arbitrary decide which solver is superior and for each model a proper one has to be chosen. A variable step solver has the advantage of dynamic adjustment of the step size according to the changes in model's states. It can accelerate when there are only small changes and reduce the step when states change rapidly. As described in III the TCP client block is not executed at every Simulink step but the solver takes care of scheduling the steps so that this block is executed at the sample time defined by the user. However, the process of choosing the suitable step size by a variable step solver may cause issues in this particular application. The solver monitors the tolerances of the states' changes and in case of violation due to a rapid unexpected change in one of the states steps back and repeats the computation with a smaller step. It can happen that it will cause the client to execute more than once for the same simulation instant. Such a case is illustrated in Fig. 7. The TCP client is executed whenever Simulink time is equal to multiple of t_{samp}. The first execution for time $t_{sim}+t_{samp}$ takes place for $t_{os}+2$ but due to violated maximal tolerance the Simulink time is put back

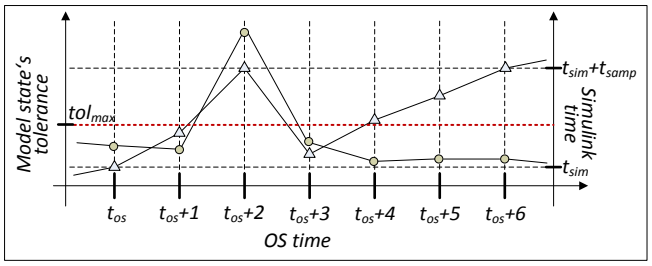

Fig. 7. Influence of the model state's tolerance on the simulink time depending on the time step. Circles represent values of model state's tolerance, triangles values of Simulink time.

and with smaller step reaches the value $t_{sim}+t_{samp}$ again at $t_{os}+6$. It could be problematic when an agent manipulates some data in external places and needs to do this in time-linear manner. On the other hand multiple performances of the whole chain for the same simulation step would slow down the co-simulation. Therefore, an additional check in the server agent was implemented in order to control the time stamp received from the client and to execute the cascade only, if the received value is greater than the last one. Additionally the user should consider definition of the maximal step size used by the variable step solver.

VI. Smart Grid Co-Simulation

The developed framework was used to design a multi-agent control system for controlling the operation of an active power grid. The agents are responsible for automated control of the MV grid to provide flexibilities at the HV/MV connection point.

A. Model of the power system

The modeled power system is based on the CIGRE MV European test system with slight adjustments [7]. Only one feeder is considered in the model, Fig. 8. Furthermore, there are additional distributed energy resources (DER) placed at MV level. One CHP is connected at node 1, a battery storage system (BSS) at node 6 and photovoltaic systems at node 5 and 9. There is also a power switch connecting the grid to the higher voltage level grid, which is considered to be the slack node in this case. The electrical part of the CHP system is modeled as a synchronous generator. The PV systems and BSS are connected to the modelled grid via ideal average model inverter with neglected switching. The control structures of these systems are modeled twofold. High level controllers are responsible for obtaining new set-points for active power, reactive power, frequency, voltage, characteristic droops. These controllers operate with higher time constants and are not suitable for fast control of the inverters. For such a control the low level controllers are used. In case of CHP these are engine governor and AVR with additional PI controllers. For inverter based systems these are only droop controllers since the inverter models do not contain valves.

B. Architecture of the multi-agent system

The high-level controllers are implemented as agents. Each DER system described in VI.A has one agent assigned. The agents communicate with low-level controllers to get measurements and to send back set-points. The PV agent requires irradiation and temperature measurement in order to be able to calculate current maximal possible active and reactive power based on the model of the PV-panel and inverter constraints stored by the agent. The BSS agent

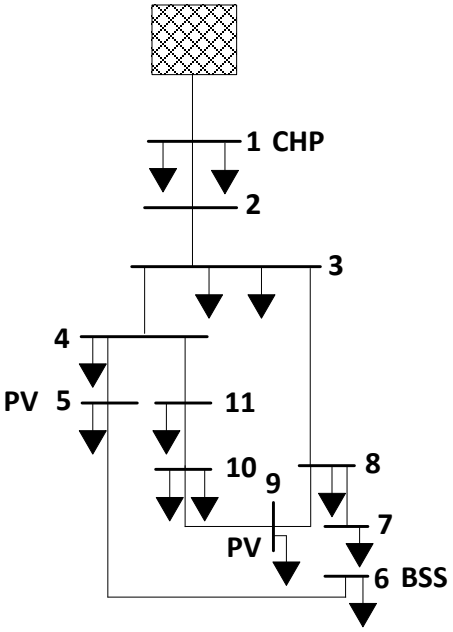

Fig. 8. Representation of the simulated power system

receives the value of SoC estimated by the low-level controller. Similarly to PV agent, the BSS agent estimates maximal possible active and reactive power which the system could provide for a given time if the grid operator would require doing so. The CHP agent does not consider fuel consumption and the only measurements is the angular velocity of the rotor and terminal voltage, mainly for sake of island operation. Additionally there is one more agent associated with the switch and responsible for keeping the power flow over the HV/MV transformer at the expected level. This agent is superior to the rest. The switch agent has access to the measurements of active power, reactive power, frequency and voltage on both sides. If the power flow varies from the scheduled it can require other agents to provide their current constraints and finds the new set-points which satisfy the requirements.

C. The framework consideration

Five agents were implemented in the described co-simulation, one for each DER and one for the switch. The TCP/IP client block had in this case 43 input and 18 output signals. The model in Simulink was simulated using variable step solver and the electric part was modeled in phasor domain. The sample time of the client block was set to 1s. Each of the five agents had periodic behaviour simulating specific measurements. A typical example of implementation of the reactive behaviour was estimation of the current constraints performed by DER agents on request of the switch agent.

VII. Summary

This paper presents a framework for integrated simulation environment for investigation of MAS in smart grid applications. It consists on interfacing a multi-agent platform developed within JADE with a model developed in MATLAB/Simulink by means of TCP/IP communication. The framework was originally developed for and tested with a power system model, however after minor adjustments can be applied to a co-simulation between JADE and Simulink in any area. Additional benefit of using TCP/IP communication for exchanging data between both environments is fact that the Simulink model can be executed on separate machine and MAS can run on distributed platform, which may significantly increase the performance. Presented example of the implementation proved usefulness of the developed solution.

For a successful implementation a set of general rules which constitute the framework have to be followed:

- Every agent who requires signals exchange with Simulink has to report this fact when registering at DF agent.
- Every agent who has time scheduled tasks (*periodic behaviour*) has to report a need of communication with Simulink no matter if exchanges any signals.
- All the signals exchanged between JADE and Simulink are alphabetically sorted by the server agent according to the agents' names given by the user. Proper distribution of the signals in Simulink is up to the user.
- Each behaviour in the used MAS has to implement the return confirmation principle. No matter if the responsible agent communicates directly with the server agent.
- The execution period of the *periodic behaviour* should correspond to the sample time of the TCP client block in Simulink.
- For the variable step solver a maximal step size should be defined.

References

[1] https://de.mathworks.com/help/physmod/sps/specialized-power-systems.html

[2] http://jade.tilab.com/

[3] Ch. R. Robinson, P. Mendham, T. Clarke, "MACSimJX: A Tool for Enabling Agent Modelling with Simulink Using JADE", Journal of Physical Agents, vol. 4, No. 3, September 2010

[4] C. J. Bankier, "GridIQ – A Test Bed for Smart Grid Agents", Master's thesis, University of Queensland, 2010

[5] R. Roche, S. Natarajan, A. Bhattacharyya, S. Suryanarayanan, "A Framework for Co-simulation of AI Tools with Power System Analysis Software", 23rd International Workshop on Database and Exper Systems Applications, September 3-7 2012, Vienna, Austria

[6] J. Gómez-Gualdrón, M. Vélez-Reyes, "Simulating a Multi-Agent based Self-Reconfigurable Electric Power Distribution System", 2006 IEEE COMPEL Workshop, July 16-19 2006, NY, USA

[7] M. Sysel, "MATLAB/Simulink TCP/IP communication", Proceedings of the 15th WSEAS international conference on Computers, July 2011, pp. 71-75

[8] https://docs.microsoft.com/en-us/windows/desktop/winsock/about-winsock

[9] F. Bellifemine, G. Caire, D. Greenwood, "Developing Multi-Agent Systems with JADE", John Wiley & Sons, Ltd, 2007

[10] K. Rudion, A. Orths, Z.A. Styczynski, K. Strunz, "Design of benchmark of medium voltage distribution network for investigation of DG integration", 2006 IEEE Power Engineering Society General Meeting, 18-22 June 2006, Montreal, Canada