A New Concept of a Generic Co-Simulation Platform for Energy Systems Modeling

Jianlei Liu, Clemens Duepmeier and Veit Hagenmeyer
Institute for Applied Computer Science
Karlsruhe Institute of Technology (KIT)
Hermann-von-Helmholtz-Platz 1
76344 Eggenstein-Leopoldshafen, Germany
Emails: jianlei.liu@kit.edu; clemens.duepmeier@kit.edu; veit.hagenmeyer@kit.edu

***Abstract*—In order to model, design, execute, control and analyze co-simulations for e.g. Smart Grids, a new scalable and generic system architecture of an agent-based co-simulation platform framework is presented in this article. Not only different kinds of simulators e.g. for power grids and technical plants, but also various types of data sources and real hardware nodes, such as wind turbines, photovoltaic cells, electrical power grid equipment, etc. which are instrumented by measurement devices, can be seamlessly integrated into the co-simulation platform in order to model large transdisciplinary, multi-domain energy systems. By integrating Apache Kafka as message exchange infrastructure into this configurable co-simulation platform the realistic simulation of SCADA communication via standard communication network protocols and services for Smart Grid including big data scenarios can be realized. As basic approaches container virtualization and microservices, namely, Docker containers and a Representational State Transfer (REST) application programming interface (API), are used as an automated runtime environment to control and manage different simulation nodes on a (larger) computing cluster. The co-simulation platform also provides an easy-to-use web browser-based user interface implemented using Angular2 to allow users to model, implement, perform and operate co-simulations for future energy system solutions without any setup or configuration on their local PC and extensive IT knowledge. Furthermore, after the completion of simulations by the individual nodes, the results of a co-simulation run can be automatically stored in databases and then analyzed and visualized afterwards via a web user interface.**

***Keywords*—Smart grid; agent-based co-simulation platform; microservice; multi-domain energy system; communication; Representational State Transfer (REST); big data**

I. Introduction

With the rapid development of science and technology, more and more electronic equipment is invented and used. Meanwhile, the demand for electricity is increasing day by day. Therefore, efficient energy production, management and usage become more and more important. Additionally, the limited availability of non-renewable resources demands an increase in using renewables for energy production. Due to the high volatility of renewable energy production, power networks with renewables need additional components, such as energy storages. They are, therefore, more complex and harder to control, which rises the need for highly sophisticated tools for the planning of modern power grids, like modern data analysis, hardware labs, and distributed (co-)simulation tools.

To foster research on new energy technology solutions for the German Energiewende the Federal Ministry of Education and Research (BMBF) and the Helmholtz Association of German Research Centers (HGF) set up the EnergyLab 2.0 and Energy System 2050 (ES 2050) research projects. The main goal of the EnergyLab 2.0 project is the development of a research infrastructure for investigating and analyzing the interactions between different components of future intelligent energy systems [1]. One central component of this research infrastructure is the Smart Energy System Simulation and Control Center (SEnSSiCC), which bundles the research on information and communication technology (ICT) for future smart energy solutions (e.g. Smart Grids). ICT related research for smart energy solutions is also the main focus of the research topic 5 "Toolbox and Data Models" of the joint research initiative ES 2050. In both projects, the KIT focuses its research on software components e.g. for the simulation of power systems, architectures for control center operation, the operation of future energy systems (data collection, management, control, data analysis, evaluation, forecasting, etc.) and further tools for planning Smart Grid solutions to improve the understanding of energy systems [2]. A co-simulation platform which allows the integration of many different simulators was identified as one very important tool for the modeling of larger and very complex future energy solutions.

There is abundant and vast literature proposing various kinds of co-simulation approaches that can be used to model, research and analyze complex systems such as Smart Grids.

As mentioned in [3], there are some well-known co-simulation interfaces standardized for large scale system model integration, e.g. High Level Architecture (HLA), Functional Mockup Interface (FMI) and Distribute Interactive Simulation (DIS), which could be used to perform co-simulations. Their main applications are in automotive industry, space and defense projects. However, the learning curve of the HLA, FMI and DIS is substantially long.

In [4] a comparison of co-simulation approaches for combining wireless sensor network models with Smart Grid application models like ns-2/adevs [5], ns-2/OpenDSS [6] and ns-2/Modelica [7] is described for analyzing the effect of SCADA communication within a Smart Grid application model. The ns-2/adevs co-simulation approach avoids some synchronization problems introduced by the coupling through integrating adevs as a module in ns-2, but this enlarges the complexity of the power behavior formalization that can drastically decrease the performance of such a platform [4]. In the ns-2/OpenDSS

coupling approach the intercommunication technique between the two simulators is not well defined [4]. The ns-2/Modelica coupling technique can be used for integrating models of wireless sensor networks with most of the existing communication technologies, such as WIMAX, WiFi, WPAN, etc. into Modelica, but Modelica can not determine when to send data to ns-2 [4]. All three mentioned approaches for the coupling of wireless communication models to Smart Grid models pose some problems.

The authors in [8] provide a communication and power distribution network co-simulation based on the event-driven simulators OMNet++ and OpenDSS for multidisciplinary Smart Grid experimentations. Another similar approach, a co-simulation framework using OMNet++ and OpenDSS for power systems and communication networks simulation, is proposed for evaluating the performance of Smart Grid wide area monitoring applications in [9]. OMNet++ and OpenDSS are running as two parallel processes and exchange events synchronized at certain time slots. But both solutions are limited to OpenDSS for grid simulations and are not easily extendible for other sorts of simulators. The authors of [10] developed a framework for co-simulations combining DigSILENT as a power system simulator and OMNET++ as a communication simulator to also simulate and analyse Smart Grid applications. Their approach is similar to the previous and only differs in the aspect that DigSILENT instead of OpenDSS is integrated into this framework and used to model power systems.

All the approaches described above combine only two simulators and are limited to particular scenarios. They do not allow the integration of many different kinds of simulators which could provide a more universal co-simulation platform for simulating real complex multi-domain energy system models combining many different model types and the associated simulators (e.g. for modeling physical components and technical plants, power and communication networks, including of economic models, or weather models).

In the German research project Mosaik [11], a more flexible Smart Grid co-simulation framework is developed, which allows users to reuse and combine existing simulation models and simulators to create Smart Grid scenarios where more than two simulators can be involved. Each simulator uses an XML-RPC (Extensible Markup Language Remote Procedure Call) frontend which allows the communications with a central coordinator and the exchange of data and events [11]. However, Mosaik does not provide tools for automatically starting and controlling simulator instances on large-scale computing clusters in order to perform simulations in Big Data context. Additionally, Mosaik does not provide real time streaming communication between the simulator nodes as needed for larger scale co-simulations with a high volume on data.

Therefore, as an important application service and planning tool of the IT infrastructure for EnergyLab 2.0 and ES 2050, a generic co-simulation platform framework is developed for modeling and simulating intelligent multi-domain energy system solutions (Smart Grids, microgrids, etc.) on the basis of large computer clusters as distributed runtime infrastructure. This solution tries to overcome the above mentioned limitations and possesses the following features to remedy the mentioned shortcomings: It runs distinct simulation models as separate processes with each encapsulating a dedicated simulator environment as runtime environment of the model by using container automation technology (e.g. Docker) on a large computing cluster. Furthermore, it provides a generic message-oriented middleware-based communication interface for data and event exchange between the different kinds of simulators, various types of data sources and real hardware nodes instrumented by measurement devices to enable an agent-based co-simulation of large complicated multi-domain energy systems. Synchronization can be based on events or - by adding time synchronization co-simulation nodes to the co-simulation environment - on virtrual clocks. In addition, the platform logic is encapsulated into a co-simulation service which can be used through an easy-to-use web user interface implemented using Angular2 just within a browser. The web user interface provides four main views: a model component editor for registering and adding models as reusable model components to the platform, a co-simulation editor for combining distinct model components to larger co-simulation models, an operator interface for running models, and a visualization and monitoring screen. The platform-independent web user interface allows different users to upload their own models as co-simulation components and to describe their runtime environment as well as their parameterization. Then, other users can combine these model components to implement, perform and run custom co-simulations of future energy system solutions. These latter users do neither need to setup nor to configure any simulation environment on their local PC since startup and management of simulation nodes is completely managed and automated by the co-simulation service and solely performed on nodes of the underlying computing cluster. Additionally, users performing modeling do not necessarily need detailed IT knowledge. Moreover, by integrating Apache Kafka and SCADA communication technology as real time message exchange infrastructure into this configurable co-simulation platform the realistic simulation of SCADA communication via standard communication network protocols and services for Smart Grid scenarios is possible, and co-simulation models can work with data coming from real field devices.

The presented article will describe basic concepts of the new co-simulation platform and is organized as follows: Section II focuses on basic information technology concepts and frameworks which are used to implement the platform. Section III provides an overview of the platform, its web user interface and the details of its implementation as well as a co-simulation example. A use case scenario will be presented in Section IV. Finally, at the end of the article a conclusion and an outlook will be given.

II. Underlying IT Concepts and Frameworks

In the following, the basic IT concepts and frameworks used to implement the co-simulation platform are described.

A. REST API

The co-simulation platform uses Representational State Transfer (REST) [1] based interfaces as lightweight communication mechanism between different components. REST is a web-based software architecture style rather than a standard.

[1] https://en.wikipedia.org/wiki/Representationa_state_transfer

It is designed to provide interoperability between computer software applications or programs across a network, such as services and web applications provided on the Internet. REST interfaces are usually based on the use of the Hypertext Transfer Protocol (HTTP) which provides the methods GET, POST, PUT and DELETE to enable communication between clients and servers. By using REST application programming interfaces (API) users can control and manage simulation models on the co-simulation platform or edit and upload model components.

B. Angular2

Angular2[2] is a development framework for building modern web applications across all operating platforms. It can provide a state-of-the-art user experience by maximizing application speed with single page application design and is scalable to meet larger data requirements by building data models on RxJS, Immutable.js or other push-model JavaScript libraries. Angular applications facilitate a component model where each component consists of a HyperText Markup Language (HTML) template and a component class that can be developed using JavaScript or TypeScript which control a part of the screen. This allows to build complex user frontends via reusable user interface and application components.

C. Apache Kafka as Message Server

Apache Kafka[3] is a message-oriented, distributed data streaming platform which is used via a Java API in Java applications for building real-time data pipelines and data processing applications. As message server environment for the co-simulation platform, Kafka can be used to exchange data between individual nodes and co-simulation services. Using the consumer group concept, Kafka generalizes the two traditional models of messaging: queuing and publish-subscribe. With the use of a queue the processing of data by the consumer group is distributed to multiple consumer instances (the different simulation members of the consumer group) to scale the processing. By instrumenting publish-subscribe, Kafka can broadcast messages to multiple consumer groups, which can process data in parallel for different purposes. In our context that means that different simulator nodes which are in multiple consumer groups can subscribe and consume the same messages in parallel (e.g. see the same state of system represented by a Kafka publish-subscribe message queue). Furthermore, by using Kafka, streams of data consisting of a key, a value and a timestamp can be stored in Kafka queue categories called topics and written to a disc for more than one week and replicated for fault-tolerance. This can be used for storing the complete data exchange within a simulation for subsequent archiving of the simulation run.

D. Docker as Runtime Environment

Docker[4] is a software containerization platform which can be used to package and isolate applications in containers by using operating system virtualization. Docker uses the separation of resources in the Linux kernel, such as cgroups and the Linux kernel namespace, to create independent software containers. Since Docker containers can contain all necessary software dependencies to execute applications, e.g. operating system and framework libraries as well as the application binaries, and furthermore, Docker images containing all these application components and dependencies can be easily transported and installed as files, the deployment of applications can be simplified and easily automated.

With the help of Docker the co-simulation platform can provide different specific Docker images to run corresponding standard-alone simulation models as separate but synchronized processes in Docker containers under the hood on the underlying computing cluster. As an automated runtime environment for running processes on a cluster, Docker can be used to operate and manage the simulation nodes and associated support services to interact with them for redefining their adjustable parameters for some complex parallel tasks such as iterative grid optimizations.

III. Basic Concepts of the Co-Simulation Platform

The following chapter discusses details of the architecture and introduces the current web user interface designed for the co-simulation platform. A simple co-simulation model with three simulators and a specific adapter for Matlab simulators will be used as example to further explain how Docker is used to manage the execution of simulation models and how the data interaction between the Matlab simulator and other components is implemented by using the Kafka message server.

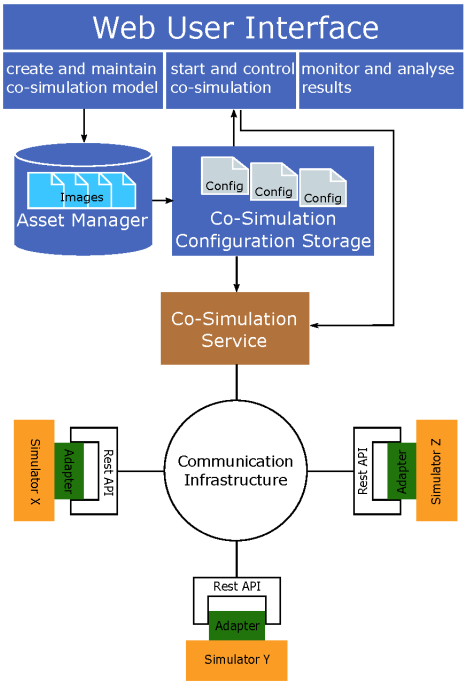

Fig. 1. The architecture of the co-simulation platform.

[2]https://angular.io/
[3]https://kafka.apache.org/
[4]https://www.docker.com/

A. Overview

The architecture of the co-simulation platform with the co-simulation service as the key component is depicted in Fig. 1. It consists of a web user interface providing several main views (see upper part of Fig. 1), the co-simulation service, the communication infrastructure and adapters to attach the simulators to the co-simulation environment.

The co-simulation service connects, starts, monitors and controls the different Docker containers running the simulators automatically based on the co-simulation configuration described by a JSON format.

Generic microservices implementing the communication infrastructure provide access to Kafka message channels for exchanging data between individual simulation nodes and data services.

Specific adapters for different simulators connect the simulators to the communication infrastructure and the co-simulation service.

B. Web User Interface

The co-simulation platform provides a web user interface allowing users to upload different simulation models as model components, define co-simulations, and operate, control as well as monitor and analyze simulation runs. The web interface is built using the Angular2 framework, HTML5 and Typescript (see Fig. 2).

Using the model component editor of the web user interface (see the left upper part of Fig. 2), not only different kinds of simulation models e.g. for modeling power grids and technical plants, but also various types of data sources and data nodes representing a data interface to real hardware nodes instrumented by measurement devices, such as wind turbines, photovoltaic cells, electrical power grid equipment etc., can be seamlessly uploaded and integrated into the co-simulation platform. Therefore, Docker images containing the software and dependencies implementing the modeling environment or the data source functionality have to be defined and described through metadata as a model component. This component can then be added to the co-simulation modeling library. Such modeling component descriptions will then be stored in the asset management system for later use by the co-simulation editor and the runtime environment.

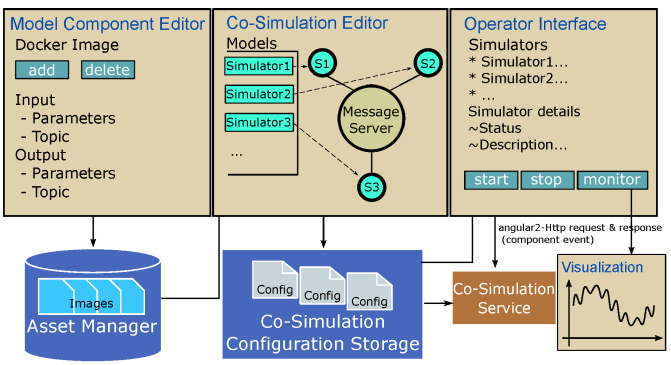

Fig. 2. The web user interface.

Co-simulations can be created by using the co-simulation editor of the web user interface (see the upper middle part of Fig. 2). On the left side of this view, available model components are listed in a vertical spreadsheet format. Users can select the associated model component with the mouse and drag them to the right side near to the message server in order to create a new node of the co-simulation. Its configuration data is persisted in a co-simulation configuration storage. The nodes can then be connected to message server queues for message exchange.

The operator interface of the web user interface (see the upper right of Fig. 2) allows users to load, start and control co-simulations. It also displays information about the co-simulation and provides a button for displaying the monitoring interface to view and visualize data exchange. After a simulation run, all data belonging to a simulation can be saved for subsequent retrieval and analysis.

C. A Simple Co-Simulation Example with three Simulators

Fig. 3 shows a small generic model with three simulators for illustration. Due to the consumer group concept of Kafka, each simulator can publish its data to one or more topic queues in Kafka and subscribe to one or more topics for receiving data from the Kafka server. Simulators in different consumer groups can also share messages of one topic. Additionally, data streams will be temporarily stored in the distributed, replicated server infrastructure of Kafka in a fault-tolerant way.

As illustrated in Fig. 3, all simulators and their specific adapters as well as the Kafka message server are packed into independent Docker containers. Since these containers include everything needed to run the simulators, such as code, runtime, libraries, etc., it is not necessary to deploy particular operating environments for different kinds of simulators. All simulators can be executed as separate but synchronized processes in their Docker containers on the computing clusters. Furthermore, by managing the containers via the Docker API the simulators can be managed and controlled simply and efficiently.

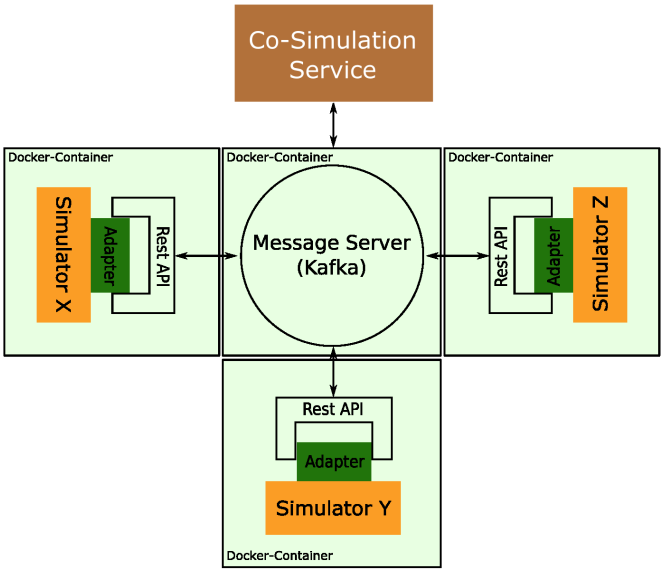

Fig. 3. A co-simulation with three simulators.

D. The Role of Adapters

In the following, the role of adapters is explained by demonstrating how a Matlab-specific adapter connects a Matlab simulation to the co-simulation communication infrastructure as shown in Fig. 4.

Fig. 4 describes how data exchange between the co-simulation communication infrastructure and a single simulator node is implemented for e.g. a single Matlab simulation node. In the specific case (a specific Matlab model) shown in Fig. 4 the communication steps needed for sending and receiving data can be broken down to:

Sending data

(1) The data to be sent (a Matlab matrix object in our example) is converted into a JSON object.

(2) Using the Matlab function ***webwrite()***, the JSON object is sent from Matlab to the co-simulation communication infrastructure adapters REST API frontend via an HTTP call.

(3) The REST API frontend then sends the JSON object to a specific Kafka topic created for this type of information. The topic stores the data on the Kafka server until all other simulators which have subscribed to this topic completed their consumption of this object.

Receiving data

(4) In our simple example, the Matlab model requests data needed from the co-simulation infrastructure by using the Matlab function ***webread()*** to call the REST API of the adapter via an HTTP call.

(5) The adapter retrieves a corresponding JSON object from the Kafka topic queue storing information of the requested type to which it has subscribed and sends it back as return value to the function ***webread()***.

(6) The Matlab simulator receives the JSON object which can then be transformed into more appropriate Matlab type(s) (e.g. a matrix object) for further use.

While writing data from the simulator node to the co-simulation infrastructure can be easily supported by a REST API Adapter if the simulator software provides some functions to send data to a third party, reading operations can be more complicated. Typically, pull operations, where the simulator model retrieves data by calling a simulator's internal function for reading data from a third party, are supported in many simulators as it is the case in our Matlab example. For pushing data that arrives in a Kafka queue as soon as possible into a running simulation on a node, the simulator environment has to support some kind of asynchronous event communication. This is often the case and the Matlab adapter will support this feature in the future.

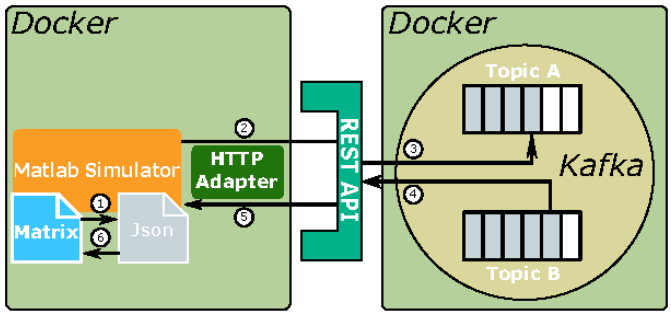

Fig. 4. Connecting a Matlab simulator to the co-simulation communication infrastructure.

While the Matlab adapter is clearly specific to Matlab, many simulators (including Matlab) support the Functional Mockup Interface (FMI) standard for model and data exchange in co-simulations. The FMI standard defines a standard C-API for data exchange which can be used together with all simulators supporting this standard. Thus, the co-simulation platform implements a generic FMI adapter that allows to connect all simulators to the co-simulation platform, which implement the FMI specification. This connects a whole bunch of simulators to the co-simulation platform in a very generic way.

IV. Simulation of a Wind Turbine as an Example of a Co-Simulation

As another more practical but simple modeling example for the usage of the co-simulation platform, a co-simulation of a wind power turbine simulating its power output in one node and providing input data (e.g. wind velocity and temperature) by a second weather simulation node or weather data source node will be demonstrated. Fig. 5 illustrates the concept of the co-simulation which consists of two simulators:

1) A data source or simulation node that generates or retrieves input data of the wind power turbine, such as pressure, wind velocity, temperature and the roughness length, and provides it to the wind turbine model (and potentially other simulation nodes) via an Apache Kafka queue with a topic named "Weather-Information"
2) A Python simulator that uses the library "windpowerlib" which is provided by the Python Software Foundation (US) to build up diverse practical wind power models and used to calculate time series of electricity production output of the wind power plant based on the weather input data [12]

Firstly, before a simulation is started, the wind turbine simulation node is parameterized by setting the values of a list of static configuration parameters which specify the height of the wind turbine and the diameter of its rotor. Similarly, the weather simulation node or weather data source node is configured to send weather data according to a certain time scheme (e.g. every 5 seconds). In our example the weather data node just pulls weather information containing already the pressure, wind velocity, temperature and the roughness length from a csv file and forwards it as JSON object to the Apache Kafka topic "WeatherInformation" of the co-simulation model. The wind turbine model exploits the data from the "WeatherInformation" topic, calculates the electrical power output and sends it to another Kafka queue with a topic named "WindTurbinePowerOutput". Therefrom the power data are extracted and stored in a database e.g. for usage in a later

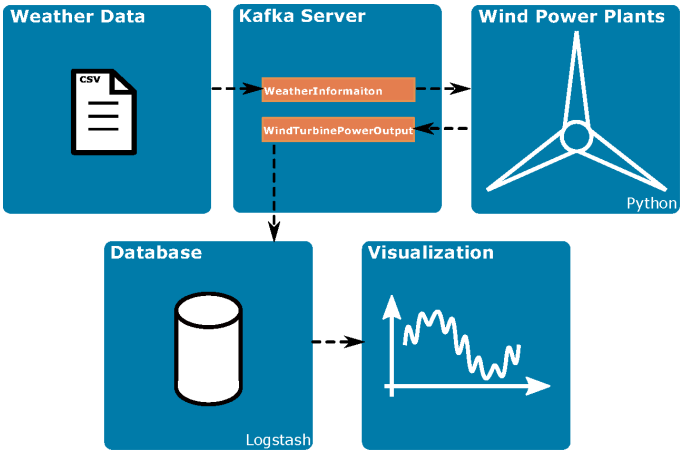

Fig. 5. A co-simulation example for a wind turbine.

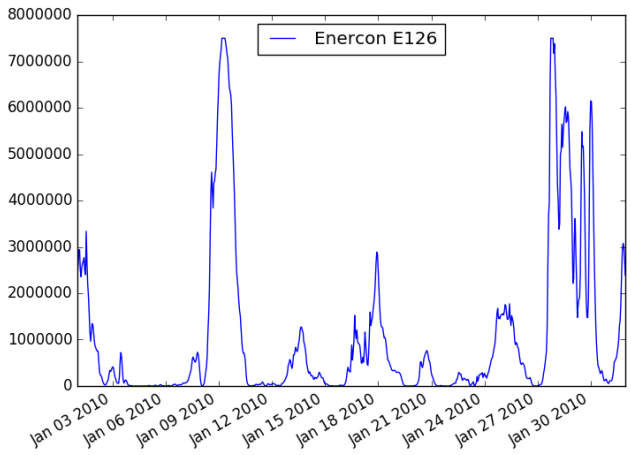

Fig. 7. A visualization of time series output of wind power plants in [W].

analysis or for visualization. Finally, a visualization application can be added to the co-simulation which monitors the power output values arriving in the database and displays them on the monitoring screen of the co-simulation operation control user interface as an appropriate visualization (e.g. a line curve).

A part of the output data stream of the wind power turbine is shown in Fig. 6. This data is converted from time series format to JSON object data and is then stored in the Kafka server before it is exported to a database. In Fig. 7, a visualization template for this scenario used by the visualization component shows an overview of power data of a wind turbine over a specific period of time in January 2010, since the weather data contained in the csv file is from 2010.

V. Conclusion and Outlook

The main scientific contribution described in this paper is the development of a new scalable and very generic system architecture for a co-simulation platform framework that is capable to model, design, plan, perform, control, analyze and evaluate co-simulations of e.g. Smart Grids. The flexibility that this co-simulation provides allows users to upload different kinds of simulators (power grid and technical plant models) and various types of data sources as well as real hardware nodes instrumented by measurement devices (electrical power grid and others) to build up and operate customized co-simulations for simulating and solving large multi-domain future energy system problems. It is also possible that users choose and combine simulators that already exist on the platform to set up a co-simulation with only using existing simulation or data source components and just tuning their configurations. As a message exchange infrastructure Apache Kafka is very scalable, high performant and intended to hide the burden of implementing the communication between different individual simulation nodes and the co-simulation service from ordinary users who just want to build co-simulation models. Through utilization of container virtualization and microservices the execution of different independent simulation nodes on the computing cluster is easily and efficiently implemented. The concrete use case mentioned in the previous chapter demonstrates the usability and extendability of the framework.

{"1262300400000":2416999.7507349062,"1262304000000":2603701.7586262776
1368798,"1262314800000":2488516.3818526366,"1262318400000":2355985.500
913.9788719569,"1262329200000":2697726.8340384872,"1262332800000":2765
0":2403104.4370905338,"1262343600000":3335448.4620151799,"1262347200000
54400000":1938489.2031359533,"1262358000000":1578849.6607214627,"12623
"1262368800000":1150686.5971671792,"1262372400000":1345196.9492548646,
711821,"1262383200000":968899.0024148548,"1262386800000":856198.794491
046544448,"1262397600000":752039.1489164013,"1262401200000":754978.686
6.0283452295,"1262412000000":237623.9556672265,"1262415600000":226840.
3828.9916481838,"1262426400000":73319.2803209329,"1262430000000":41355
957.2583422441,"1262440800000":42257.0254598311,"1262444400000":75037.
0064.3081577245,"1262455200000":254604.7703479355,"1262458800000":3378

Fig. 6. Time series output of the wind power turbine in [W].

In the future, a larger modeling library has to be integrated into the framework for building up complicated co-simulations, which will instrument other simulation environments, such as Modelica tools, Simulink, OpenDSS, DigSilent (power factory), PSLF (positive sequence load flow), etc., in order to further improve the generic applicability of the platform. The web user interface consisting of the parameters settings panel for models can be enhanced to allow users to dynamically adjust the value of parameters in order to obtain different co-simulation results. A further future improvement could lie in an even better and universal access to background data by fully instrumenting the microservice-based data management platform of the EnergyLab 2.0. This will allow models to dynamically and ceaselessly read data they need and swiftly calculate the relevant output.

References

[1] V. Hagenmeyer, H. K. Cakmak, C. Duepmeier, T. Faulwasser, J. Isele, H.B. Keller, P. Kohlhepp, U. Kuehnapfel, U. Stucky, S. Waczowicz, and R. Mikut. Information and communication technology in energy lab 2.0: Smart energies system simulation and control center with an open-street-map-based power flow simulation example. *Energy Technology*, 4(1):145–162, 2016.

[2] C. Duepmeier, K. Stucky, R. Mikut, and V. Hagenmeyer. A concept for the control, monitoring and visualization center in energy lab 2.0. energy informatics. *Smart Grid Modeling and Simulation, IEEE First International Workshop*, 9424:83–94, 2016.

[3] C.-H. Yang, G. Zhabelova, C.-W. Yang, and V. Vyathkin. Co-simulation environment for event-driven distributed controls of smart grid. *IEEE Transactions on Industrial Informatics*, 9(3):1423–1435, August 2013.

[4] S. Chouikhi, I. E. Korbi, Y. Ghamri-Doudane, and L. A. Saidane. A comparison of wireless sensor networks co-simulation platforms for smart grid applications. *International Journal of Digital Information and Wireless Communications*, 2013.

[5] A disc rete event system simulator. http://web.ornl.gov/~1qn/adevs/; visited Januray, 11th 2017.

[6] T. Godfrey, S. Mullen, R. C Dugan, C. Rodine, D. W. Griffith, and N. Golmie. Modeling smart grid applications with co-simulation. *The 1st IEEE International Conference on Smart Grid Communications*, pages 29–296, 2010.

[7] V. Liberatore and A. Al-Hammouri. Smart grid communication and co-simulation. *IEEE Energytech conference*, 2011.

[8] M. Levesque, D. Q. Xu, M. Maier, and G. Joos. Communications and power distribution network co-simulation for multidisciplinary smart grid experimentations. *NSERC Strategic Project Grant*, 2011.

[9] K. Angappan D. Bhor and K. M. Sivalingam. A co-simulation framework for smart grid wide-area monitoring networks. *IEEE*, 2014.

[10] I. Ahmad, J. H. Kazmi, M. Shahzad, P. Palensky, and W. Gawlik. Co-simulation framework based on power system, ai and communication tools for evaluating smart grid applications. *Innovative Smart Grid Technologies-Asia*, 2015.

[11] S. Schuette, S. Scherflke, and M. Troeschel. Mosaik: A framework for modular simulation of active components in smart grids. *The series Lecture Notes in Computer Science*, pages 55–60, 2011.

[12] A python library to create time series from wind power plants. https: //pypi.python.org/pypi/windpowerlib/; visited Januray, 20th 2017.