Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 00 (2018) 000–000

www.elsevier.com/locate/procedia

12th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 18-20 July 2018, Gulf of Naples, Italy

Synchronization of a “Plug-and-Simulate”-capable Co-Simulation of Internet-of-Things-Components

Tobias Jung*, Michael Weyrich

University of Stuttgart, Institute of Industrial Automation and Software Engineering, Pfaffenwaldring 47, 70550 Stuttgart, Germany

* Corresponding author. Tel.: +49 711 685-67301; fax: +49 711 685-67302. *E-mail address:* ias@ias.uni-stuttgart.de

Abstract

Modern production systems are increasingly interconnected and flexible and therefore form Internet-of-Things-systems (IoT). Because of the flexibility those systems are also dynamic, meaning the entering and leaving of components during runtime, and heterogeneous. For the simulation of such systems, those challenges of dynamic and heterogeneity have to be met. Therefore the authors presented an agent-based co-simulation concept, but an important aspect of a co-simulation is the synchronization of the used simulations, which hasn’t been considered yet. In this contribution the challenges of synchronizing a co-simulation of IoT-systems are introduced and existing co-simulation synchronization concepts examined with regard to their usability for simulating IoT-systems. Afterwards a synchronization concept is presented, which can be used in the presented agent-based co-simulation concept.

Keywords: Internet fo Things systems; co-simulation; agent-based modelling; synchronization; Digital Twin

1. Introduction

Due to the introduction of the Internet of Things technology, automated systems in production and logistics get more and more dynamic and heterogeneous. IoT-components can enter or leave the system during runtime and through the interconnection across different domains, like production, logistics, etc., the components of IoT-system differ greatly. To meet these challenges, which occur during every phase of a life-cycle, new concepts, like “Plug-and-Play” during operation are presented. For the simulation of such IoT-systems also new concepts are needed to meet the challenges of dynamic and heterogeneity. A possible approach is “Plug-and-Simulate”.

Reference [1] investigates existing approaches and concepts for the simulation of IoT-systems and evaluates them with regard to their “Plug-and-Simulate”-capabilities. It is concluded, that a novel co-simulation concept is needed. In this concept each IoT-component is simulated in its own simulation environment and each simulation is represented by a software agent. The single simulation are therefore coupled by the resulting agent system and can interact with each other via agent communication, whereby the communication in the real IoT-system is simulated. The representation by agents enables the simulations to enter and leave the co-simulation during runtime. As different simulation tools are used to meet the challenge of heterogeneity, translators between the simulation tools and their respective agents are needed, to translate the messages of the simulation tools to the, by the agents used, exchange format. This concept is shown in Fig. 1.

Currently only sequential or time-in depended scenarios can be simulated with the presented concept. But if parallel scenarios, which occur frequently in IoT-systems, have to be simulated and when using different simulation tools, as done in the presented concept, the problem of synchronizing the different simulations arises. An example for a scenario, which needs synchronization, is the simulation of an IoT-based warehouse, where forklifts, which are transporting goods, are simulated. The simulation of a forklift is triggered by an incoming good, which is then transported to a storage area. During the transportation of the good the forklift gets a second message by another forklift that it has to change its route, to prevent collisions with other forklifts. If the simulation of the first forklift is much faster than the simulation of the second forklift, the second message is received, when the first forklift has already reached its destination, whereas it should have be received while driving. Such errors in co-simulation are called causality errors. Hence the different simulation tools have to be synchronized. All in all the question arises, how "Plug-and-Simulate"-capable co-simulations can be synchronized.

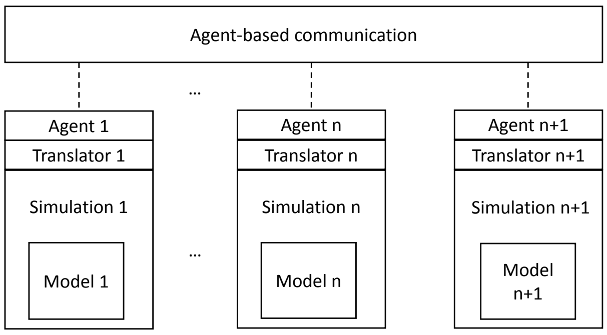

Fig. 1. Multi-agent-based co-simulation.

Therefore, at first in chapter 2, the basics of the synchronization of co-simulations are presented, to make basic decisions for the needed concept. Afterwards synchronization concepts of existing co-simulation approaches and standards are researched and evaluated with regard to their "Plug-and-Simulate"-capabilities. In chapter 3 basic design decisions for a synchronization of "Plug-and-Simulate"-capable co-simulations are presented. Chapter 4 presents a concept for the synchronization of the simulation tools in the presented agent-based co-simulation concept. In the end, in chapter 5, a description of a prototypical implementation is given, which can be used for an evaluation of the presented concept.

2. Existing Synchronization Concepts for Co-Simulations

2.1. Advancing Time in Co-Simulations

Before synchronization in co-simulations can be realized it has to be determined how the simulators advance their logical time during the simulation. In [2], three possibilities of advancing time in a simulation are presented:

- Event Driven Time Advancing: The simulation itself is a discrete event simulation and the simulation steps are divided into events. In the simulation those events are executed one after another in the correct order. This order is mostly derived from time stamps belonging to the events.
- Time Stepped Time Advancing: The simulation itself can be a continuous or a discrete event simulation and the simulation is divided into time steps. Those time steps can either have a fixed or flexible duration and all calculation for a time step have to be finished by all simulations before advancing to the next time step.
- Wall Clock Driven Time Advancing: The simulation itself can be a continuous or a discrete event simulation. The time advancement depends not only on the execution speed of the simulation itself, but also on an external wall clock, which runs continuously. Real time requirements can be met with this time advancing method.

Depending on the used time advancing mechanisms the synchronization of a co-simulation can be simplified. For example if only "Wall Clock Driven Time Advancing" is used, no time stamps are needed, but then the co-simulation is not time efficient. If "Time Stepped Time Advancing" is used, before advancing to the next time step, it must be guaranteed, that all messages belonging to this time step were received. In case of using "Event Driven Time Advancing" it has to be guaranteed that no events of the past are received. Here the problem can arise, that the durations and granularity of events in different simulation tools can differ, which is why, different guaranteeing methods have to be applied than when using "Time Stepped Time Advancing".

A simplification of the synchronization concept can only be made, if it can be guaranteed, that one or more of the presented time advancing methods will not be used in a co-simulation. To meet the above mentioned heterogeneity of the IoT-systems a huge variety of simulation tools will be utilized, which is why, this cannot be guaranteed and all time advancing mechanisms have to be considered for the synchronization.

2.2. Conservative and Optimistic Synchronization

Traditionally synchronization methods are divided into conservative and optimistic synchronization [3].

Conservative mechanisms guarantee, that no event is processed out of turn, in case of discrete event simulation and that no message is delayed or received too late in case of continuous simulation. Therefore either, in case of continuous simulations, a central instance regulates the advancement of the simulations and only allows an advancement, when all current messages are received and all current events are processed or, in case of discrete event simulation, a mechanism has to be implemented, that guarantees, that all events are delivered in the correct order. Several mechanisms and algorithms for a conservative synchronization exist and can be classified into the methods "synchronous operation", "with dead-lock avoidance", "deadlock detection and recovery" and "conservative time windows" [4]. Common for all those methods is, that every event and every message requires a time stamp to guarantee the avoidance of causality errors.

Optimistic synchronization algorithms allow causality errors and have the ability to detect them. For the detection of causality errors timestamps can also be used, but also other ways to detect them exist. If a causality error in a simulation was detected, the simulation has to be rolled back, meaning that all preceding simulation results have to be undone until the causality error is resolved. Before the occurrence of a causality error the simulations of a co-simulation are not synchronized and run therefore independently of each other and are only synchronized in case of a causality error. For realizing an optimistic synchronization, all previous inputs of the simulation have to be saved, as they are needed again in case of a roll back. Additionally a mechanism to call back previous outputs has to be installed, as in case of a causality error those outputs can also be wrong. One of the best known optimistic synchronization algorithms is the "Time Wrap algorithm" by Jefferson [5].

A benefit of the optimistic synchronization mechanisms is, that the simulation time can be greatly reduced, as the individual simulations can all run in their own time and do not have to wait regularly for slower simulation. In the best case scenario even the shortest possible execution time for the co-simulation can be achieved. However, the resource utilization is higher than with conservative synchronization methods because not all calculated results are useable.

Because of the reduction of the execution time an optimistic synchronization is desirable, but there also much higher requirements on the simulation tools, as they have to be able to store input as well as output values and have to be able to roll back the simulation time. In case of conservative synchronization, the simulation tools only need the ability to pause its simulation at discrete points in time or between discrete events. Therefore the useable synchronization methods highly depends on the used simulation tools.

2.3. Synchronization in Functional Mock-up Interface (FMI)

According to [6] FMI is not capable of "Plug-and-Simulate". Nevertheless, it is a popular co-simulation concept used in the automotive sector and therefore its synchronization concept was investigated, whether some aspects can be adapted to the presented agent-based co-simulation concept.

In FMI two possibilities for co-simulation exist. One possibility is the coupling of subsystem models where the individual simulations (simulation slaves) are encapsulated in so called Function Mock-up Units (FMUs), which then are integrated into a single simulation tool acting as the simulation master. Here the FMUs contain both the model and a solver that is able to execute the model. The other possibility is tool coupling, where the models are executed in their native simulation tools, acting as simulation slaves, which are coupled via FMI wrappers to the simulation tool acting as the simulation master [7].

The simulation master is responsible for both the data exchange between the simulations and the synchronization of the simulations. The most commonly used synchronization methods in FMI are conservative methods, more specifically "synchronous operation"-methods, meaning in case of FMI the data exchange between the simulations happens at discrete points in time. These discrete points are called communication points and all simulation slaves stop their simulation at each communication point. Then the simulation master collects all outputs of the simulation slaves and distributes them to the corresponding simulation slaves as inputs. At each communication point the simulation master also provides the current simulation time to each simulation and the size of the next simulation step, meaning the time until the next communication point. These step sizes can either be constant or can vary for each simulation step [8].

If a conservative synchronization method is used for a "Plug-and-Simulate" co-simulation, the synchronization concept of FMI can be used, but the simulation tools, used for the co-simulation, have to meet some requirements. All of them have to able to stop their simulations at fixed points in time and therefore must be able to have a continuous flow of simulation time, or, if they do not provide a continuous flow of simulation time, they have to be able to subdivide events, which take longer than the set simulation step size. If the concept is extended by the possibility, that not all of the simulation tools have to be stopped at every communication point, but that some only participate at some communications points, the number of useable tools would increase. The main drawback of this concept is its time efficiency, as all of the simulations have to wait at each communication for the slowest simulation to be finished.

2.4. Synchronization in High Level Architecture (HLA)

HLA is in principle capable of "Plug-and-Simulate" [6], even though only known models and simulation tools can be integrated during runtime. A typical HLA co-simulation consists of the simulations, called federates, a Run Time Infrastructure (RTI), as well as several specifications. The RTI is coordinating the data exchange and synchronization of the co-simulation and is comparable to a simulation master, but is not a simulation itself. Those specifications are needed to connect the federates to RTI and to enable a data exchange [9].

For HLA both conservative (time step advancement and event-based advancement) and optimistic synchronization methods exist. To participate in the synchronization, federates have to send Time Stamp Order (TSO) messages, which contain the time stamp when the message was sent. Additionally those messages can contain a Lookahead value, which is the time in which a federate will not send any TSO messages, starting from the time the last TSO message was send. By sending a Lookahead value it can be guaranteed, that no other messages will be sent in a certain amount of time and the other federates can advance their time for that time period. If a federate has a time-based advancement, it has to make a Time Advance Request at the RTI, which is only granted, after the RTI has checked, whether the conditions for a time advancement are met. For event-based advancement it works in a similar way, only that a "Next Message Request" has to be made to the RTI. If an optimistic approach is used, Message Retraction methods have to be applied. When a federate receives a message with a lower logical time than its own simulation time, it has to send Request Retraction message to the other federates to retract the wrongly send messages. However, the detection of causality errors has to be handled by the federates themselves [10].

All in all, synchronization methods used for HLA co-simulations vary, depending on the used RTI and simulation tools, but most of the available synchronization methods are "Plug-and-Simulate" capable. Those methods all have their own benefits and drawbacks, mostly similar ones as previously mentioned.

2.5. Synchronization of further Co-Simulation Approaches

Additionally to FMI and HLA other co-simulation approaches, which are to some degree capable of "Plug-and-Simulate", were researched. In [11] OPC UA was utilized for co-simulation. Here each simulation is connected via an interface to a generic adapter consisting of an OPC UA server and an OPC UA client. The different simulations can communicate with each other via an aggregating server. Whereas [12] uses OSGi to couple different simulation, by integrating each participating simulation into an OSGi-Bundle to enable a communicate between the simulations via the OSGi framework. However, for those approaches, no information about the used synchronization concept were available.

3. Synchronization of "Plug-and-Simulate" capable Co-Simulations

In principle, both FMI and HLA synchronization methods, as well as the above mentioned conservative and optimistic methods can be used for "Plug-and-Simulate"-capable Co-Simulations. But they can only be used without restriction, if all used simulation tools support those methods, which leads to a small number of useable tools, as many simulation tools have huge limitations regarding the possibilities to influence the simulation time during runtime.

For many simulation tools it is difficult to pause a simulation during runtime or to execute a simulation in discrete time steps with variable step sizes. Most simulation tools are not capable of rolling back a simulation. Therefore the synchronization method with the least requirements to the simulation tools, the conservative method with "synchronous operation", is most suited, although an optimistic approach would be desirable in regards to the overall simulation time. This results in a synchronization method for "Plug-and-Simulate"-capable co-simulations, where every simulation is paused after each time step. Those time steps have to be variable. If the time steps are not variable, there is always the possibility that the current time step size cannot be executed by a new simulation tool that enters the co-simulation.

This approach still leaves open the possibility to extend the synchronization by rollback mechanisms, if the rollback is handled by the simulation tools themselves or by their representing agents, although not all simulations will be able to participate in rollback.

It is still possible that during one time step several messages are send and it cannot be guaranteed by the agent system that those are received in the correct order. Therefore the time steps have to be sufficiently small, so that it is either not possible, that several messages can be send and received during a single time step, or so that it does not matter in which order the messages are received during a time step, as it can be assumed, that they were received at the same time. As for many scenarios neither of those two options is practicable, a message exchange between the simulations is only allowed at the end of each time step, which also requires sufficiently small time steps.

The size of the time steps depends on the simulated scenario and therefore has to be set for each co-simulation. To simplify the use of "Plug-and-Simulate", the smallest possible time step for the co-simulation will be chosen by default when a new simulation enters the co-simulation. The smallest possible time step is the least common multiple of the smallest possible time steps of each simulation.

4. Synchronization of an agent-based co-simulation

4.1. Multi-Agent System for Co-Simulation

As, because of the above mentioned reasons, a conservative synchronization method without rollback mechanisms was chosen, a central instance is needed to coordinate the synchronization. Therefore an additional "Clock Agent" besides the agent representing the simulations is introduced. During a co-simulation, each simulation executes a time step and then reports to its representing agent, that the simulation step is finished. Each agent sends, upon receiving this report by its simulation, a message to the "Clock Agent", that the current simulation step is finished. When all agents have reported, that their current simulation step is finished, the "Clock Agent" sends a message to each representing agent to start the next simulation step. The concept with the added "Clock Agent" (grey) can be seen in Fig. 2.

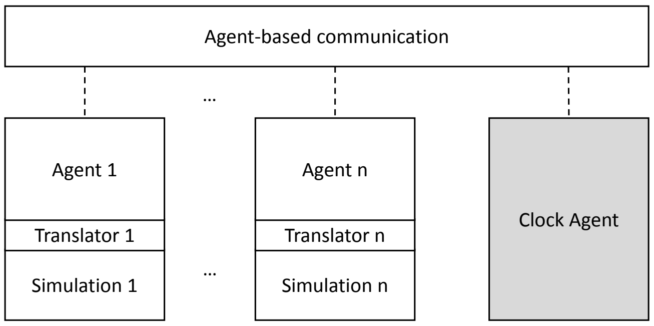

Fig. 2. Synchronized, multi-agent-based co-simulation.

Instead of using an additional agent it is also possible to use an external clock, running in a parallel system to the agent system. A benefit of such a parallel system is, that such a system can meet real time requirements, in contrast to an agent system, where real time cannot be guaranteed, as agent system are not always deterministic. Therefore such an external clock can provide a much more exact time stamp, which is especially needed for "Wall Clock Driven Time Advancing". But such a parallel system increases the complexity of the interface of the simulation tools, as an interface to both the agent system and the clock system is needed. And if no real components are included in the simulation systems, as in Hardware-in-the-Loop simulations, and the time steps are sufficiently small, the time provided by a “Clock Agent” is exact enough. Therefore the concept of the “Clock Agent” was chosen.

Another problem arises, when a “passive” simulation participates in the co-simulation. “Passive” means that a simulation is in an idle state, unless it is triggered to generate a reply. An example would be the simulation of a temperature sensor, which only measures the temperature, when a measurement request was posed and is otherwise idle. Such a simulation will not keep track of its simulation time, while it is in its idle state. Therefore, it cannot reply to the “Clock Agent” that it has finished the current time step. In those cases, the agent will reply directly to the “Clock Agent”, that its simulation has finished, when the message to start the next simulation step arrives and while the simulation is in its idle state. For this an agent always assumes, that upon receiving a message the simulation will switch to an active state and will remain there until the reply has been generated. Therefore the reply message has to contain, in addition to the actual reply, the information, whether the simulation has switched to idle after sending the reply or not, so the agent knows whether to wait or to send an immediate reply after receiving the next message to start the next simulation step.

Therefore, in addition to the “Clock Agent”, the agents representing the simulations have to be extended by the reply mechanisms and the ability to identify, whether a simulation is idle.

4.2. Interface between Agents and Simulation Tools

For forwarding the synchronization messages, the agents need an interface to the simulation tool. This interface is in parallel to the already existing translator interface, responsible for the forwarding of the messages of the IoT-system.

The translator interface is divided into a generic and a tool specific part. The generic part is connected to an agent and is the same for all simulations to increase reusability. The specific part is connected to a simulation tool and has to be adapted for each simulation tool, as each simulation tool has its own specific interface to interact with other programs. This concept is adapted for the synchronization interface, dividing the synchronization interface also into a generic and a specific part.

The generic part receives the messages from the “Clock Agent” to start the next simulation step and forwards it to the specific part. It also reports to the “Clock Agent”, that the current simulation step was completed. The specific part will start or resume the simulation for each new simulation step and therefore needs a connection to the simulation tool. It will also receive the information when the simulation finished the simulation of the time step, either by getting notified by the simulation tool, if possible, or by regularly checking the simulation, whether the time step is finished. The specific part differs for each simulation tool, but can be reused for different models and scenarios in the same simulation tool. The specific part also has to manage the setting of the time steps. As the step size can always change, when a new simulation enters the co-simulation. Then, the step size has to be sent together with the message, which starts the next time step.

The extended concept of the interface between agents and simulations, including the translator and the synchronization interface, can be seen in Fig. 3.

Additionally an interface is needed to add the information to messages exchanged between the simulations, whether the simulation is idle or active. This interface needs both a connection to the specific translator, as it has to extract the information about the idle state from the reply of the simulation to the former request and needs additionally a connection to the specific synchronization interface, to which it has to pass this information (Fig. 3). After extracting the information about the idle state of the simulation, the interface deletes this part of the message, as it should not be passed to the other simulations, because they cannot utilize this part of the message. The information about the idle state will be passed to the generic part of the synchronization interface and to the agent of the simulation, where it will be stored, so the agent can always reply to the message by the “Clock Agent” to start the next time step.

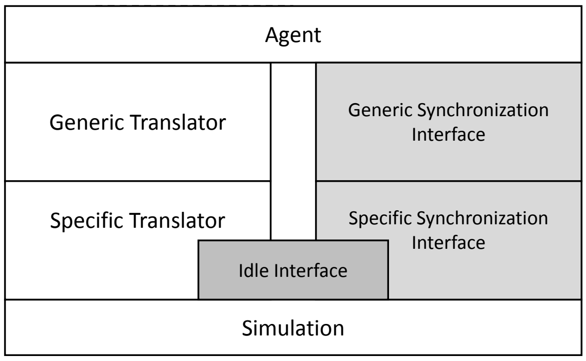
Agent
Generic Translator
Generic Synchronization Interface
Specific Translator
Specific Synchronization Interface
Idle Interface
Simulation

Fig. 3. Interface between Simulation Agents and Simulation Tools.

5. Prototypical Implementation

Currently the concept is not completely implemented, the synchronization interface is only partly implemented and the “Clock Agent” is not realized yet. The models and the agent system on the other hand exist already and the “Plug-and-Simulate”-capabilities can be shown with this prototype.

5.1. Scenario and chosen Tools

For the prototype a scenario of an IoT based temperature and humidity controlling system was chosen, consisting of models of a temperature sensor, a humidity sensor, a heating unit and a humidity controller. The heating unit and the humidity controller request the values of their respective sensor and start heating or respectively increasing the humidity, when the values fall under a certain threshold. Additionally to the models of the IoT components a model of the environment is needed, as the heating unit and the humidity controller can also interact via the environment with the sensors, by heating or changing the humidity. To simulate those interactions, “physical messages” are sent to and from the environment model. For example, if the heating unit is heating, it tells the environment, that it is currently heating with 20 kW. If the heating unit is turned off again, it sends a message to the environment, that it stopped heating. In case that the heating unit and the humidity controller are turned off, the environment model decreases continuously the temperature and the humidity.

The sensor models are discrete models, which are idle most of the time, unless a measurement was requested. The models of the heating unit and the humidity controller are idle, when turned off, but continuously simulating when turned on and the environment model is a continuous model all of the time. Therefore, both continuous and idle models are included in the scenario. A synchronization is needed between the environment model and the models of the heating unit and the humidity controller, when turned on. This is necessary because those models are continuously running and without a synchronization the point in time, when the heating unit and the humidity controller are turned off again could be missed by the environment model. Additionally “idle messages” have to be sent by the sensor models and the models of the heating unit and the humidity controller.

The models of the environment and the sensors are done in MATLAB/Simulink and the models of the heating unit and the humidity controller are done in OpenModelica.

The agent system itself was implemented based on Jadex, a Java based framework for developing multi agent systems.

5.2. Interface between Agents and Simulation Tools

To implement the synchronization interface, especially the specific part, knowledge about the used simulation tools is needed, basically, how the simulation time of the simulations can be influenced during runtime.

MATLAB/Simulink has in principle the possibility to run a simulation step by step and the interval of the steps can be adjusted. But this possibility primarily only exists for the user interface and not for command lines, which is needed to run and pause the simulation via the specific part of the synchronization interface. Though, if MATLAB/Simulink is executed in debug mode, it is possible to run the simulation step by step. To vary the step size, an additional command has to be used.

OpenModelica itself offers no possibility to run a simulation step by step. But as OpenModelica can be used as a simulation slave in a FMI co-simulation, the interface provided by FMI can be used to run and pause the simulation.

6. Summary and Outlook

This contribution extends an already existing agent-based co-simulation concept, which is capable of “Plug-and-Simulate”, by a concept for the synchronization of the different participating simulations. For this the following aspects were described:

- At first, the need for the synchronization of simulations in co-simulations was shown.
- Afterwards, the basic concepts and methods of synchronizing simulations, like conservative and optimistic methods, were researched and evaluated, with regard to their performance and their usability for “Plug-and-Simulate”
- Then already existing co-simulation concepts and approaches were researched and afterwards evaluated, especially with regard to their capability to employ “Plug-and-Simulate”.
- Afterwards, those concepts were adapted for synchronizing an agent-based co-simulation and the concept of “idle messages” was introduced.
- In the end a description of a scenario for evaluating the presented concept and a prototypical implementation, which is already partly realized, was given.

The next step will be to finish the realization of the prototypical implementation to evaluate the concept presented. Afterwards the concept will be extended by more sophisticated synchronization methods to improve the overall simulation time of the co-simulation.

References

[1] T. Jung, N. Jazdi, and M. Weyrich, “A survey on dynamic simulation of automation systems and components in the Internet of Things,” in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, pp. 1–4.

[2] R. M. Fujimoto, “Time Management in The High Level Architecture,” *Simulation*, vol. 71, no. 6, pp. 388–400, Aug. 1998.

[3] R. M. Fujimoto, “Parallel discrete event simulation,” *Commun. ACM*, vol. 33, no. 10, pp. 30–53, 1990.

[4] S. Jafer, Q. Liu, and G. Wainer, “Synchronization methods in parallel and distributed discrete-event simulation,” *Simul. Model. Pract. Theory*, vol. 30, pp. 54–73, Jan. 2013.

[5] D. R. Jefferson and D. R., “Virtual time,” *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 3, pp. 404–425, Jul. 1985.

[6] M. Oppelt, G. Wolf, and L. Urbas, “Capability-analysis of co-simulation approaches for process industries,” in *ETFA'2014*, 2014, pp. 1–4.

[7] et alt T. Blochwitz, “The Functional Mockup Interface for Tool Independent Exchange of Simulation Models,” in *The 8th Modelica Conference*, 2011, pp. 105–114.

[8] Modelica Association Project “FMI,” “Functional Mock-up Interface for Model Exchange and Co-Simulation,” no. 07006, pp. 1–120, 2013.

[9] “IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-- Framework and Rules.” IEEE, Piscataway, NJ, USA, 2010.

[10] O. Topçu, Ed., *Guide to distributed simulation with HLA*. Cham, Switzerland: Springer, 2017.

[11] S. Hensel, M. Graube, L. Urbas, T. Heinzerling, and M. Oppelt, “Co-simulation with OPC UA,” *Proceedings, 2016 IEEE 14th International Conference on Industrial Informatics (INDIN.Palais des Congrès du Futuroscope, Futuroscope - Poitiers, France, 19-21 July, 2016*. IEEE, Piscataway, NJ, p. 20–25 TS–CrossRef, 2016.

[12] M. Oppelt, O. Drumm, B. Lutz, and A. G. Gerrit Wolf Siemens, “Approach for integrated simulation based on plant engineering data,” in *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, 2013, pp. 1–4.