Distributed co-simulation for software-in-the-loop testing of networked railway systems

Tobias Pieper
University of Siegen
Institute for Embedded Systems
Siegen, Germany
Email: tobias.pieper@uni-siegen.de

Roman Obermaisser
University of Siegen
Institute for Embedded Systems
Siegen, Germany
Email: roman.obermaisser@uni-siegen.de

***Abstract*—Software-in-the-loop (SIL) is a common methodology for repeatable and controllable component testing during the development process of complex systems. Testing the integration of distributed embedded systems is accomplished on a network centric abstraction level. However, todays state-of-the-art is missing a solution for the railway domain which couples local simulations via the Internet or LANs to support those tests between companies.**

The framework proposed in this paper is based on the simulation standards FMI (Functional Mock-up Interface) and HLA (High Level Architecture). They are combined with further mechanisms for configuration, delay-management, fault-injection, monitoring and simulation control. This solution enables SIL testing via the Internet without sharing any software. Hence, it facilitates the interplay between different railway manufacturers and allows early integration tests wherefore testing effort and time can be reduced compared to today.

I. Introduction

During the development process of complex systems, Software-In-The-Loop (SIL) is a common methodology for repeatable and controllable component testing. This simulation type does not require physical prototype systems wherefore it supports early validation and avoids damage of real prototype systems. The integration and testing of train components are important development steps in the railway domain. These steps can be improved compared to today's state-of-the-art using a distributed simulation and validation framework which operates on a network centric abstraction level. Such a framework couples local simulations via the Internet or LANs to provide distributed SIL simulation.

Many simulation tools are designed for special purposes, for example communication networks or controlled plants. Coupling such specialized tools to interact for simulating complex systems is called co-simulation. In contrast to existing railway solutions, our proposed distributed co-simulation framework supports SIL testing without the need of centrally available simulation models. Hence, it is possible to test pieces of software which are located at different railway manufacturers or locations via the Internet in early development stages.

The network centric abstraction level enables the connection of train devices as a whole and to inject faults on network level to test the devices' behavior in case of faulty inputs. As shown in this paper, a framework covering all those aspects is not available today. In our framework, simulation bridges synchronize the attached simulations and exchange information between them. The simulation bridges provide a generic interface to support various simulation tools and communicate with each other using heterogeneous and public communication networks such as the Internet. Since the framework shall be generic to support testing of various train applications, it requires a mechanism for configuration. A configuration interface enables the initial configuration before the simulation starts and reconfiguration at dedicated points during runtime. In combination with a control interface, the reconfiguration further provides automated test execution.

The remainder of this paper is organized as follows. Section 2 presents the current state-of-the-art in co-simulation and SIL testing, while Section 3 discusses requirements of the framework. In Section 4, the framework's high-level design is described. Section 5 concludes the paper.

II. Related works

This section focuses on related works about SIL testing and the concept of co-simulation. Furthermore, it presents different co-simulation solutions such as the High Level Architecture (HLA) as well as the Functional Mock-up Interface (FMI).

A. Software-in-the-loop testing

SIL is a widely used approach to validate the interaction of a software control algorithm with the model of a simulated plant [6]. This technique enables the verification of the system during the design phase taking into account technological side-effects. One important effect is time-to-market, hence system components are developed independently and integrated in a later step. Potential integration problems between them can be avoided by verifying their interactions in early development stages. SIL further enables the analysis of the system's behavior in case of faults [6].

In various domains, SIL is already used for fast prototyping of control systems as well as electronic and mechatronic devices [18]. Examples can be found in the automotive domain [26][19][27], avionics [24] and other domains [6]. Demers et al. [11] point out weaknesses of traditional modeling and simulation. Among others, model abstractions may lead to a loss of design details and the introduction of errors. Hence, simulation validation is difficult which results in a high time

exposure to build a valid model. Furthermore, the usability of model code for the software implementation might be constrained. SIL can solve those problems since there is no need for a model if the final software is used directly. The authors further denote scalability, the need to modify software for connecting it to existing co-simulation solutions, the breaking of TCP connections while rerouting network traffic to a network simulation tool and timing issues during the synchronization process as remaining challenges.

B. The co-simulation concept

Co-simulation frameworks couple specialized tools to simulate complex systems in cooperation. Since porting components between simulators is time consuming and error prone, the tools are connected via a framework [8]. The framework synchronizes the simulation tools which run in their own processes. Thereby the tools maintain their internal state and their time which might differ between them [12].

Typically, discrete-event-based simulators are coupled with continuous-time-based ones. Continuous differential equations defining the system dynamics and the transition between state variables are discretized. In addition, the time base is divided into small steps so that there are no transitions of system variables within a step [17]. The selection of a time-step in a discrete-event tool is difficult. While small steps waste simulation time if the state remains unchanged, long steps might miss events. Hence, the tools hop between the events based on a chronologically ordered list managed by a scheduler. Such a heterogeneous environment requires a synchronization and communication mechanism which must ensure a message delivery in the correct order (consistency) and at the earliest next time step (on-time) [17]. In a simple algorithm, the discrete-event tool defines a step granularity. It sends the step to the continuous tool which advances its time until the step has finished or an event occurred. In case of an event it notifies the discrete tool and provides all required data such as monitored variables and the internal simulation time. The discrete tool can now advance to the same time [12].

There are various co-simulation solutions based on communication via TCP/IP sockets in LANs or WANs. Examples are an interface between MATLAB and OPNET [14], a solution coupling OPAL-RT and OPNET [3] or the co-simulation of Gem5 and OPNET [22]. Two co-simulation standards the developed framework uses are presented in the next section.

C. The High Level Architecture and the Functional Mock-up Interface

The High Level Architecture (HLA) connects individual components to a set called federation. One component (a federate in terms of the HLA) might be a computer simulation, a supporting utility or an interface to a live partition [10]. The HLA was designed independently from any language or platform and supports solutions to the most common problems of interoperability [2]. It imposes no constraints on what is represented, it only requires specified capabilities for the interconnection with other federates. These interactions are realized by the exchange of data.

Data exchange is implemented as services in the Runtime Infrastructure (RTI) which acts as a distributed operating system. Besides data exchange, it provides basic services for the management of objects, time and data distribution. The services are accessible using a defined API which is available in different languages, e.g. C++. Each federate must document its object model using a standard object model template. This model facilitates information sharing and reusability [10].

The Functional Mock-up Interface (FMI) is a tool-independent standard for model exchange or the co-simulation of models [5]. For this it provides an interface which is implemented by various simulation tools. FMI for model exchange aims at the creation of a modeling environment which is able to generate C-code of a dynamic system model. This code can be used by another simulation environment as input/output-block. FMI for co-simulation establishes a co-simulation environment where multiple simulation tools (slaves) are coupled by a master algorithm. The master synchronizes the slaves and exchanges data at discrete communication points. Between these points, the subsystems are solved independently [4].

Components which implement the FMI standard are called Functional Mock-up Units (FMUs). They consist of one zip-file including an XML-file, the model and additional data. The additional data is optional and can be documentation files or utilized libraries. The XML-file contains the definition of all environmental variables as well as other information about the model. The model is deployed as C source code or in binary for different platforms. If the model is used for co-simulation, it contains functions (I) for the initiation of the communication with a simulation tool, (II) to compute a communication time step and (III) to exchange data. Otherwise, the model includes only the model equations [5].

In todays state-of-the-art, various master algorithms are available. They focus on deterministic execution of FMUs [7], the co-simulation of continuous and discrete dynamics [9] or performance improvements by controlling the communication step size [25]. Although FMI supports different master algorithms, it does not define one in the standard [9]. Awais et al. [2] propose the usage of the RTI as a generic master algorithm. Their implementation shows great opportunities of integrating both standards and the possibility of using them to realize a heterogeneous, distributed simulation platform. Garro and Falcone [13] investigate the combination of the standards from two perspectives, (I) HLA for FMI and (II) FMI for HLA. For each perspective, they provide solutions in their work. Neema et al. [21] identify challenges of integrating FMUs for co-simulation via the HLA. Their approach uses different solvers and step sizes to overcome non-linearity as well as discontinuities. Although the HLA provides means for interoperability, the integration of simulations can be difficult. Hence the authors use the HLA as master algorithm for FMI.

D. Research gap

Executing simulations on geographically distributed machines connected via LAN or WAN provides advantages like (I) parallel execution, (II) project decentralization, (III) design by different development teams and testing of the system directly at the manufacturer and (IV) resource sharing [1]. However, the network introduces delays which might increase the simulation execution time. Examples for distributed simulation environments are presented by Amory et al. [1], Kelley et al. [16] or Hopkins et al. [15].

The previous sections show various solutions for SIL and co-simulation. However, none of them covers the full set of requirements described in the next section. On the one hand, the solutions are developed for different domains such as automotive or avionics or do not cover a network centric simulation of railway components. On the other hand, the co-simulation solutions are limited to specific simulation tools if no changes in the communication and synchronization mechanisms are possible. Even if the approach of Demers et al. [11] is a generic solution for network simulation, its setup in a LAN and the synchronization mechanism using broadcasts is not suitable for distributed simulation via the Internet.

The limitations can be solved using the framework presented in this paper which is based on the combination of the HLA and FMI. While FMI supports various simulation tools, the HLA provides the advantages of distributed co-simulation. Extensions further cover the remaining requirements.

III. Requirements of the Distributed Simulation Framework

The design of the distributed simulation framework is based on the following requirements which cover co-simulation, applicability and configuration.

A. Co-simulation requirements

One main requirement is the possibility to connect simulation tools via the Internet or LANs to support the distributed co-simulation of different devices in a train. The underlying communication system shall be based on layers 2-4 of the OSI model and the framework shall be executable on Windows and Linux to support a wide range of manufacturers. Therefore the mechanism to synchronize the simulations and to exchange data between them needs to be independent from any operating system. The simulation execution shall be controllable by an operator wherefore an interface is required to start and stop it. During the simulation execution, there shall be possibilities to inject faults into the messages and to monitor the behavior of all devices.

B. Applicability requirements

To provide easy and high applicability, existing simulation models and tools shall be re-usable by providing a generic interface in the framework. The subsystems of the simulation framework shall be modular and contain defined interfaces. This facilitates the integration of future techniques.

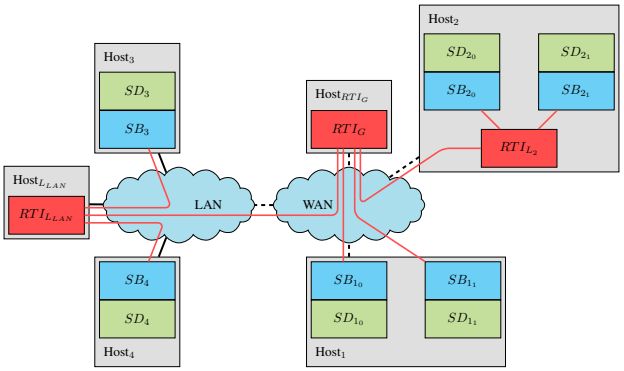

Fig. 1. Architecture of the Distributed Simulation Framework.

C. Configuration requirements

Since the simulation framework shall be usable in a wide range of applications, a mechanism to configure its subsystems is required. Therefore, each subsystem must contain an interface by which the simulation parameters can be set. The configuration information shall be available as data file with a defined and human readable structure. Integrity and coherency errors shall be detected and reported to the user before the simulation starts. During runtime, it shall be possible to reconfigure the systems using the same mechanism.

IV. High-Level Architecture

This section consists of two parts. First, the overall architecture of the distributed simulation framework is presented. Afterwards, an explanation of the simulation bridges' architecture follows. This explanation includes the behavior of each subsystem during the simulation execution.

A. Overall architecture

Figure 1 presents the overall architecture of the distributed simulation framework. The main components are Simulation Hosts (printed as gray boxes), instances of the HLA's Runtime Infrastructure (RTI, red boxes), simulation bridges (SB, blue boxes) and Simulated Devices (SD, green boxes).

Simulation hosts can be PCs, servers or clusters. They are interconnected by topologies based on LANs (solid lines) and WANs (dashed lines), hence most of todays networks can be used to run a simulation. There are various RTI implementations available such as the commercial MÄK or Pitch RTIs or the open source solution OpenRTI. The framework shall be independent from any RTI implementation and since OpenRTI covers all services required, it will be used.

OpenRTI is a hierarchical solution where each RTI instance runs as own process. In Figure 1, three instances are shown. RTI_G (global RTI) denotes the process on the highest level in the hierarchy which connects the local RTIs in the LAN and on Host2. Using such a hierarchy, network load is reduced since the RTI instances only forward subscribed interactions. This enables scalability in simulations with multiple, distributed hosts. A central RTI would represent a bottleneck in the execution of such a simulation.

Logically, the simulation bridges operate on the Data Link Layer of the OSI stack. From the devices perspective, a simulation bridge performs the same operations like a bridge in a real network. It wraps packets from the device into an RTI interaction and includes additional data such as time-stamps for delay management. This interaction is then published and transmitted to all subscribing simulation bridges. Those decapsulate the initial Ethernet packet, analyze the temporal behavior to manage delays and forward the message to the device. Moreover, the simulation bridges synchronize the advance in time of all simulations using the HLA time management services. They ensure the sending of all interactions in the correct temporal order (called Time Stamped Order) by restricting time-advances of the federates.

The tools simulating the devices are executed on a host and connected to a simulation bridge via FMI. It is possible to run one simulation per host (see Host3 and Host4) or multiple ones (see Host1 and Host2). Multiple simulations on the same host can be connected to a local RTI instance (see Host2) or to a remote one (see Host1). Which possibility is used depends on simulation requirements and the available hardware resources.

B. Architecture of simulation bridges

The architectural model of the simulation bridges is shown in Figure 2. It contains subsystems for co-simulation, state-estimation, delay-management, fault-injection, configuration, monitoring and a wrapper to connect the device. Furthermore, there are buffers for input and output packets, estimated ones as well as forwarded packets with faults introduced (Forward Packet Buffer).

The co-simulation subsystem connects the simulation bridge to an RTI process by setting up a simulation execution and joining it. Before a simulation run starts, it performs the publish/subscribe process for interactions and unpublishes/unsubscribes them before disconnecting. At runtime, it exchanges data with other bridges and synchronizes the devices using an algorithm based on the NextMessageRequest-service of the HLA. This service advances the logical time of a federate to either the requested time or a time before when interactions are received. Before it notifies the federate about the time advance grant, it forwards all subscribed interactions which are available. The co-simulation subsystem inserts the received interactions into the input packet buffer and notifies the delay-management subsystem about their availability. It further provides an interface to start and stop the simulation.

Since the Internet can be used for the communication, there might be undetermined delays. In future work, also Hardware-In-The-Loop testing shall be included. Here, real-time requirements need to be covered which requires the management of delays. The delay-management subsystem calculates the communication delay between the bridges using time-stamps. It terminates the simulation execution if the delays are too large and the simulation results would be degraded. To handle delays and to increase the performance, inputs for the device can be estimated in the state-estimation subsystem. For this, the subsystem uses a model of the entire system and injects

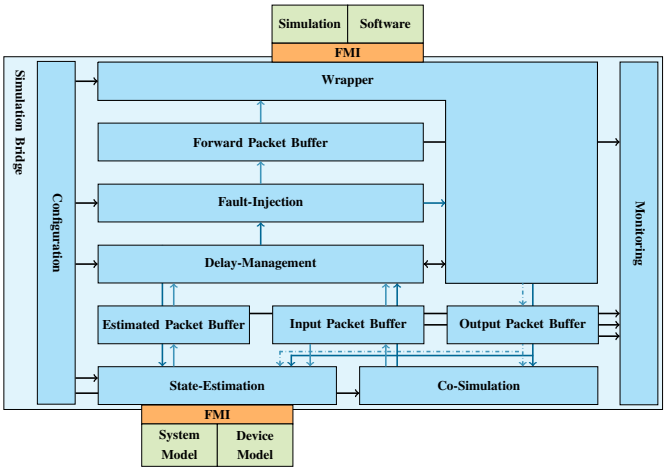

Fig. 2. Architecture of the simulation bridges.

the estimated inputs into the estimated packet buffer. Based on an event schedule, the delay-management subsystem forwards the estimated packets between the received ones. To ensure the quality of the estimation, the state-estimation subsystem further compares the estimated state of the device with the one based on the packets received. For this, a model of the device is connected besides the system model.

One of the main purposes of this framework is testing the communication behavior of networked railway systems. Faulty communication needs to be handled by the devices which is why the framework supports the injection of faults according to the EN 50159 standard for safety in train applications. The fault-injection subsystem injects faults according to its configuration and inserts the packets into the forward packet buffer as they will be forwarded.

The device is connected to the wrapper subsystem using FMI for co-simulation. FMI provides a set of API (Application Programming Interface) calls to set interface variables and to control the execution of a simulation step. The packets are represented as interface variables and set by the wrapper subsystem before the step to the granted time is performed. After the step finished, the wrapper subsystem gets the packets from the device and checks the destination IP addresses to encapsulate the packet into an interaction. This interaction is inserted into the output packet buffer and the wrapper subsystem signals the co-simulation subsystem about its availability. The co-simulation subsystem then publishes the interaction.

Observing the system behavior is enabled by the monitoring subsystem. It has access to the buffers and is able to get the monitoring information for each interaction. This information includes, e.g., the message identifier, its payload, temporal and fault information and is written into a file. After the simulation has finished, all monitoring files can be analyzed by the user.

The configuration subsystem is responsible for configuring all subsystems in the simulation bridge. It reads the configuration information and analyzes it with respect to errors in integrity and coherency. If the information is accepted, the subsystem sets the related parameters in the other subsystems.

During the execution, the configuration subsystem can be used further to reconfigure the system at dedicated points. This enables the support of automated simulation execution with different test cases.

V. Conclusion

Integrating and testing train components are important development steps in the railway domains. Compared to today's state-of-the-art, these steps can be improved using a distributed simulation framework which supports SIL simulation via the Internet or LANs.

Our presented approach concentrates on SIL testing without the need of centrally available simulation models. A combination of the HLA as master algorithm for FMI is extended by mechanisms to manage delays, inject faults as well as configure and monitor the subsystems. Future works will concentrate on the implementation and the evaluation of the design presented. Furthermore, additional services will be included to provide Hardware-In-The-Loop testing together with SIL.

Acknowledgment

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730830.

References

[1] A. Amory, F. Moraes, L. Oliveira, N. Calazans, F. Hessel. "A heterogeneous and distributed co-simulation environment [hardware/software]", Integrated Circuits and Systems Design. Proceedings. 15th Symposium, pp. 115-120. IEEE, 2002.

[2] M. Awais, P. Palensky, A. Elsheikh, E. Widl, M. Stifter. "The high level architecture RTI as a master to the functional mock-up interface components". In Computing, Networking and Communications (ICNC), 2013 International Conference on, pp. 315-320. IEEE, 2013.

[3] D. Bian, M. Kuzlu, M. Pipattanasomporn, S. Rahman, Y. Wu. "Real-time co- simulation platform using opal-rt and opnet for analyzing smart grid performance". In 2015 IEEE Power & Energy Society General Meeting, pp. 1-5. IEEE, 2015.

[4] T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, et al. "Functional mockup interface 2.0: The standard for tool independent exchange of simulation models". In Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany, number 076, pp. 173-184. Linkping University Electronic Press, 2012.

[5] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, H. Elmqvist, A. Junghanns, J. Mau, M. Monteiro, T. Neidhold, D. Neumerkel, et al. "The functional mockup interface for tool independent exchange of simulation models". In Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical University; Dresden; Germany, number 063, pp. 105-114. Linkping University Electronic Press, 2011.

[6] C. Bonivento, M. Cacciari, A. Paoli, M. Sartini. "Rapid prototyping of automated manufacturing systems by software-in-the-loop simulation". In 2011 Chinese Control and Decision Conference (CCDC), pp. 3968-3973, 2011.

[7] D. Broman, C. Brooks, L. Greenberg, E. Lee, M. Masin, S. Tripakis, M. Wetter. "Determinate composition of fmus for co-simulation". In Proceedings of the Eleventh ACM International Conference on Embedded Software, p. 2. IEEE Press, 2013.

[8] S. Ciraci, J. Daily, J. Fuller, A. Fisher, L. Marinovici, K. Agarwal. "Fncs: A framework for power system and communication networks co-simulation". In Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative, p. 36. Society for Computer Simulation International, 2014.

[9] F. Cremona, M. Lohstroh, S. Tripakis, C. Brooks, E. Lee, 2016. "Fide: an fmi integrated development environment". In Proceedings of the 31st Annual ACM Symposium on Applied Computing, pp. 1759-1766. ACM, 2016.

[10] J. Dahmann, R. Fujimoto, R. Weatherly. "The dod high level architecture: an update". In Simulation Conference Proceedings, 1998. Winter, volume 1, pp. 797-804. IEEE, 1998.

[11] S. Demers, P. Gopalakrishnan, L. Kant. "A Generic Solution to Software-in-the-Loop". MILCOM 2007 - In IEEE Military Communications Conference, pp. 1-6, 2007.

[12] J. Fitzgerald, P. G. Larsen, M. Verhoef. "Collaborative Design for Embedded Systems: Co-modelling and Co-simulation". SpringerLink : Bcher. Springer Berlin Heidelberg, 2014.

[13] A. Garro, A. Falcone. "On the integration of hla and fmi for supporting interoperability and reusability in distributed simulation". In Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, pp. 9-16. Society for Computer Simulation International, 2015.

[14] C. Harding, A. Griffiths, H. Yu. "An interface between matlab and opnet to allow simulation of wncs with manets". In 2007 IEEE International Conference on Networking, Sensing and Control, pp. 711-716. IEEE, 2007.

[15] K. Hopkinson, X. Wang, R. Giovanini, J. Thorp, K. Birman, D. Coury. "Epochs: a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components". IEEE Transactions on Power Systems, 21(2):548-558, 2006.

[16] B. M. Kelley, P. Top, S. G. Smith, C. S. Woodward, L. Min. "A federated simulation toolkit for electric power grid and communication network co-simulation". In Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2015 Workshop on, pp. 1-6. IEEE, 2015.

[17] H. Lin, S. S. Veda, S. S. Shukla, L. Mili, J. Thorp. "Geco: Global event-driven co-simulation framework for interconnected power system and communication network". IEEE Transactions on Smart Grid, 3(3):1444-1456, 2012.

[18] M. Malvezzi, E. Meli, S. Papini, L. Pugi. "Parametric models of railway systems for real-time applications". In Multibody dynamics, Milano, Italy, 2007.

[19] K. Mitts, K. Lang, T. Roudier, D. Kiskis. "Using a Co-simulation Framework to Enable Software-in-the-Loop Powertrain System Development". SAE Technical Paper 2009-01-0520, doi:10.4271/2009-01-0520, 2009.

[20] M. N. Mladenovic, M. M. Abbas. "Multi-scale integration of Petri Net modeling and Software-in-the-loop simulation for novel approach to assessment of operational capabilities in the advanced transportation controllers". In 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC), pp. 1051-1056, 2001.

[21] H. Neema, J. Gohl, Z. Lattmann, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Batteh, H. Tummescheit, C. Sureshkumar. "Model-based integration platform for fmi co-simulation and heterogeneous simulations of cyber-physical systems". In Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden, number 096, pp. 235-245. Linkping University Electronic Press, 2014.

[22] Z. Owda, M. Abuteir, R. Obermaisser. "Co-simulation framework for networked multi-core chips with interleaving discrete event simulation tools". In 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), pp. 1-8. IEEE, 2015.

[23] P. J. Roache, P. Knupp M. "Completed richardson extrapolation". Communications in Numerical Methods in Engineering, 9(5):365-374, 1993.

[24] P. Royo, C. Barrado, E. Pastor. "ISIS+: A Software-in-the-Loop Unmanned Aircraft System Simulator for Nonsegregated Airspace". In Journal of Aerospace Information Systems, Vol. 10, No. 11, pp. 530-543, 2013.

[25] T. Schierz, M. Arnold, C. Clau. "Co-simulation with communication step size control in an FMI compatible master algorithm". In Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany. No. 076. Linkping University Electronic Press, 2012.

[26] Q. Yan, J. Williams, J. Li. "Chassis Control System Development Using Simulation: Software in the Loop, Rapid Prototyping, and Hardware in the Loop". SAE Technical Paper 2002-01-1565, doi:10.4271/2002-01-1565, 2002.

[27] M. Zoppi, C. Cervone, G. Tiso, F. Vasca. "Software in the loop model and decoupling control for dual clutch automotive transmissions". In 3rd International Conference on Systems and Control, Algiers, pp. 349-354, 2013.