Network-Centric Co-Simulation Framework for Software-In-the-Loop Testing of Geographically Distributed Simulation Components

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*—Distributed embedded real-time systems are typically composed of several subsystems which have to be integrated and tested. The components are integrated with real-time communication networks and need to provide reliable and timely services to the system's physical environment. Co-simulation and Software-In-the-Loop (SIL) testing are common methods to simplify the development process. In SIL testing, implementations of software components are coupled with a simulation model of the remaining system. Testing the software components against the simulation model enables the early identification of design faults and helps to avoid integration problems in later development stages. If a single simulation tool is unsuited for all parts of the remaining system (e.g., continuous physical environment, communication networks, other software components), then co-simulation can be used to connect specialized simulation tools.**

However, in many cases the simulation models for the different parts of the system are geographically separated or simulation models cannot be shared due to intellectual property protection. Therefore, a co-simulation framework for SIL testing supporting the coupling of geographically distributed simulation components is required. In this paper we introduce a distributed co-simulation framework for SIL testing which operates at a network-centric abstraction level. Using the framework, the components can be verified without integrating all simulation models and software components at a central place. The framework is validated based on a fault-tolerant fan control application. We experimentally evaluate the framework by comparing the simulation performance of local, LAN-based and Internet-based simulations.

I. Introduction

Integrating and testing components are important steps in the development process of distributed embedded systems. An early validation of a software component against other software components and simulation models helps to prevent integration problems in later development steps. Both, the software components and the simulation models can be located at different organizations. To protect their intellectual properties [15], these organizations might not intend to share their models or software implementations. However, the components are integrated with real-time communication networks. This enables component testing on a network-centric abstraction level without knowledge about the component's implementation.

Software-In-The-Loop (SIL) testing is a widely used approach for repeatable and controllable component testing of complex systems. Since a software application is coupled with the model of a simulated plant, a physical prototype is not required. Therefore, early validation is supported and there is no risk to damage a real prototype. Many simulation tools are designed for dedicated purposes, e.g. the simulation of networks or system dynamics. Their coupling to design complex systems in cooperation is called co-simulation. Furthermore, co-simulation can be used to connect geographically distributed simulation tools using a convenient framework.

In today's state-of-the-art, there is a lack of frameworks for SIL testing of geographically distributed simulation components. Therefore, [24] proposes requirements and a concept of a distributed co-simulation framework. The framework enables SIL testing without the need of centrally available simulation models and software components. This way the simulated devices or software under test can be located directly at the manufacturer.

This work introduces the simulation bridges which are the main part of the proposed framework besides the High Level Architecture's (HLA) Runtime Infrastructure (RTI). The implementation focuses (I) on realizing a synchronization algorithm based on the HLA's time management services, (II) the exchange of data between the simulation bridges and (III) the connection of devices via the Functional Mock-up Interface (FMI). Furthermore, we present a fault-tolerant fan control application which we use to evaluate the performance of the framework. Here, we define three topologies, a local setup, a setup where the simulation bridges and an RTI instance communicate via a heterogeneous network as well as a hierarchical setup. We analyze the simulation durations, the interaction delays between the simulation bridges and the time a packet needs to pass through the bridges.

The remainder of this paper is organized as follows. Section 2 discusses related works in co-simulation and SIL testing, while Section 3 presents the most important aspects of our implementation. In Section 4 we depict the fault-tolerant fan control algorithm as well as the topologies mentioned above in detail. After analyzing the evaluation results in Section 5, we conclude the paper in Section 6.

II. Related works

This section presents related works covering the concept of co-simulation and software-in-the-loop (SIL) testing. It presents two co-simulation standards, the High Level Architecture (HLA) and the Functional Mock-up Interface (FMI), and analyzes different frameworks used in the railway domain.

A. Co-simulation and software-in-the-loop testing

Simulating devices is a common mechanism to test their behavior in early development steps. In today's state-of-the-art there are many simulation tools which are optimized for the respective domains. Examples are the simulation of plants or communication networks. However, these tools have disadvantages if they are not used for their intended purpose. The concept of co-simulation solves these issues by coupling simulation tools using a framework. This framework enables a cooperative simulation of complex systems [9]. Each tool runs as process maintaining its internal time and state while the framework realizes the synchronization [14].

There are two types of simulation tools, discrete-event and continuous-time-based simulators. In continuous-time-based tools, the system dynamics are defined using continuous differential equations and transitions between state variables. Both need to be discretized and the continuous time needs to be divided into small steps. In those steps no transitions take place [23]. The selection of a time-step in a discrete-event tool is based on a chronologically ordered list of events. During the execution, the tool hops between these events. A heterogeneous environment with both simulation types therefore requires a synchronization mechanism which advances the simulations correctly in time. To realize the co-simulation, the simulation tools typically provide message-based communication interfaces. Hence, a communication mechanism must further ensure the delivery of messages at the earliest next time step and in the correct order [23].

Validating the interaction between the model of a simulated plant together with a software-implemented control algorithm is called software-in-the-loop (SIL) [5]. The system can already be verified during the design phase which prevents potential integration problems in later development steps. Another advantage is the usability of the final software during testing. Building a valid model is time consuming, design details are lost due to abstraction and errors might be included. Building a model is not required if the software is used [12]. Today, SIL is used for fast prototyping in various domains such as automotive [27][19][28] or avionics [25]. However, there is no satisfying solution that concentrates on a distributed, network-centric co-simulation for SIL testing.

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

The ground for this simulation framework is the combination of two simulation standards, the High Level Architecture (HLA) and the Functional Mock-up Interface (FMI). Both can be used in different domains such as military, public transportation and the development of control systems [20].

In the HLA, components (federates) such as computer simulations are connected to a federation [11]. Since its design does not depend on any language or platform, the HLA provides solutions to many common interoperability problems [2]. The only requirement is the support of specified capabilities to interact with other federates. This interaction is supported by exchanging data via messages.

The main component in the HLA is the Runtime Infrastructure (RTI) which acts similar to a distributed operating system. It provides services to manage objects, the advancement of time and to distribute data. Each service can be accessed via a defined API which is implemented in different languages such as C++. A standard object model facilitates information sharing and reusability. Hence, each federate must document its model using a defined template [11].

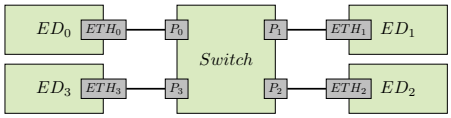
Besides a standard for co-simulation, FMI provides one for the exchange of simulation models. By providing a generic interface which various tools implement already, FMI is independent from any simulation tool [4]. FMI for model exchange creates an environment containing C-code of a dynamic system model. This environment can be used by another simulation tool as input/output-block. FMI for co-simulation couples multiple simulation tools (slaves) by a master algorithm. The master synchronizes the tools and exchanges data at discrete communication points. Between these points the simulations are solved independently [3].

A component implementing the FMI standard is called Functional Mock-up Unit (FMU). FMUs consist of an XML-file providing model information and the code of the model. Furthermore, it may contain optional data such as documentation or additional libraries. The model code contains the model equations and functions to (I) establish the communication with a simulation tool, (II) compute a communication step and (III) exchange data [4].

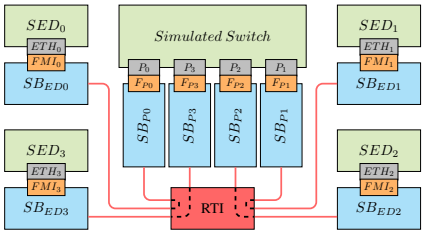
FMI does not define a master algorithm in the standard [10]. However, there are various algorithms available which focus on different aspects. Examples are performance improvements based on controlling the communication step size [26] or the deterministic execution of FMUs [6]. While both algorithms are explicitly designed for FMI, Awais et al. [2] suggest to use the HLA as generic master. Their integration shows significant opportunities in integrating the standards to realize a distributed, heterogeneous platform.

C. Research gap

Although various solutions for SIL testing exist already, there is a lack of frameworks connecting geographically distributed simulation components. The presented frameworks are either centralized, only support dedicated simulation tools or do not concern the network-centric abstraction level. This also accounts for distributed simulation environments such as those presented by Amory et al. [1], Kelley et al. [18] or Hopkinson et al [16]. Demers et al. [12] present a distributed approach which uses broadcasts to synchronize the simulation tools. Hence, it is only suitable for simulations in a LAN and not via the Internet.



(a) Distributed device under test (DUT).



(b) DUT connected via the simulation framework.

Fig. 1. Usage of the simulation framework. (a) Device under test with real communication links. (b) Links between the subsystems are replaced by pairs of simulation bridges. Red lines denote connection via LAN or Internet.

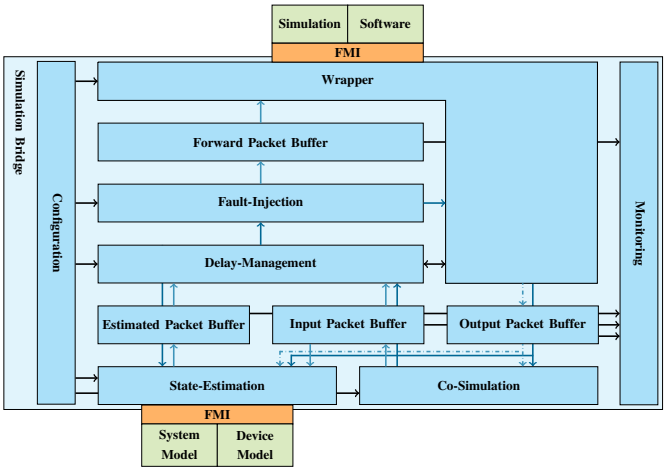


Fig. 2. Architecture of the simulation bridges.

In contrast to the works mentioned before, the HLA is designed for distributed simulations. Ficco et al. [13] use the HLA to integrate a large number of different simulation tools and real-time prototypes. Their middleware realizes time synchronization and communication between local or distributed tools and prototypes. However, it does not concentrate on the network-centric abstraction level we focus on. Furthermore, simulations are modeled as federates which requires a high effort and much code to be added. Our emphasis is a co-simulation framework which allows the easy integration of geographically distributed simulation tools and software applications.

III. The Distributed Simulation Framework

The following section describes the architecture of the simulation framework and its main components, the simulation bridges. It further explains the mechanism to synchronize the connected simulations and software applications as well as the handling of packets from both directions.

A. Architectural overview

Our distributed simulation framework focuses on testing the software components and simulation models on a network-centric abstraction level via the Ethernet protocol. The wire used to connect two devices is replaced by two simulation bridges. The bridges work on the Data Link Layer of the OSI stack in a transparent way. They capture the Ethernet frames from the sending device and forward them to the receiving one. Figure 1 shows the connection of the simulation bridges and the DUT. Assume the DUT consists of four end devices which are wired to a four-port switch via Ethernet (Figure 1 (a)). Examples for those end devices may be a distributed, simulated plant and a related controller. Using the framework (Figure 1 (b)), we replace each black, physical link by two simulation bridges (blue boxes). They communicate with the RTI (red box) via a heterogeneous communication network such as a LAN or the Internet using the TCP protocol (red lines). In the RTI, the physical links are denoted as black, dashed lines to indicate the connections between each pair of simulation bridges.

As explained before, the simulations need to be synchronized by a master algorithm which also provides the data exchange. To reach this, our solution uses the data distribution and time management services of the HLA. They are provided by the Runtime Infrastructure (RTI) as the standard's central component. It is available as commercial implementation (MÄK or Pitch RTI) or as open-source (OpenRTI). Since the OpenRTI provides all services required, it is used for the evaluation in this work. The HLA is a distributed simulation standard using TCP/IP communication. Hence, it is possible to run the simulations on a local host via sockets, in a LAN or via the Internet.

Figure 2 shows the main building blocks of the simulation bridges. A wrapper connects the simulated device using the FMI standard for co-simulation, the fault-injection subsystem injects faults according to the EN 50159 standard and the delay-management manages delays introduced by the communication network between the bridges. To manage the delays, the state-estimation module estimates future input packets based on previous outputs from the device and a model of the remaining DUT. The delay-management module decides if it has to forward an estimated or a received input packet to the higher layers. This decision is performed according to the configured communication schedule of the application. Packets from other simulation bridges are received by the co-simulation module. It further sends data from the device and synchronizes the bridge with the other bridges in the simulation setup. To access the packets, the simulation bridge contains four buffers. These are four buffers for input and output packets, estimated ones as well as forwarded packets with faults introduced (Forward Packet Buffer). They can be accessed via the monitoring module to analyze the data.

B. Synchronization and time management

Synchronization is reached using the HLA services for time management [17]. These services can guarantee a consistent

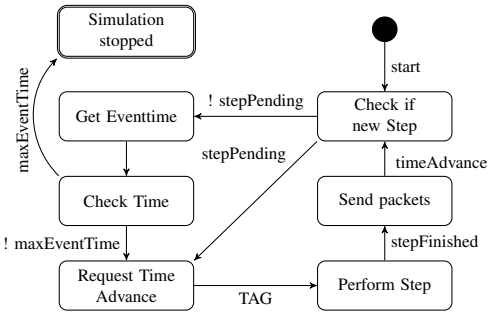


Fig. 3. Main loop in the co-simulation module.

message delivery throughout an entire federation. In our framework, the federation is the set of simulation bridges and the devices connected to them.

Time is represented as points along a time axis. A federate can associate itself and its activities with points on this axis which are then denoted as the federate's logical time. The association of activities is established by assigning timestamps to the messages sent. There are two types of messages, timestamp-ordered (TSO) or receive-ordered (RO) ones. While a receive-ordered message can always be sent or received, sending and receiving timestamp-ordered messages is constrained by the RTI. To send this message type, the federate must be time-regulating and assign the sending time to the message. On the other hand, only time-constrained federates can receive timestamp-ordered messages.

During the simulation execution, the time-regulating federates control the advancement of the time-constrained federates' logical time. The federates use one of the RTI's services to advance in time, e.g. *time advance* or *next message request*. As parameters, these services contain a timestamp which denotes the time until which the federate guarantees not to send a timestamp-ordered message. The RTI uses these timestamps to calculate an upper bound ensuring that no time-constrained federate will receive a timestamp-ordered message in its past. Only if the bound advances beyond the federate's requested time, the RTI grants the time advance. In this case, the RTI also transmits all messages destined to the federate.

The co-simulation module uses the *next message request* service to advance in time. In contrast to *time advance request*, the RTI already grants a time advance to a time before the requested one if a timestamp-ordered message is received. This way, the simulation bridge is able to react on events which were not foreseen in the configuration. If *time advance request* is used, the granted time is always the requested one.

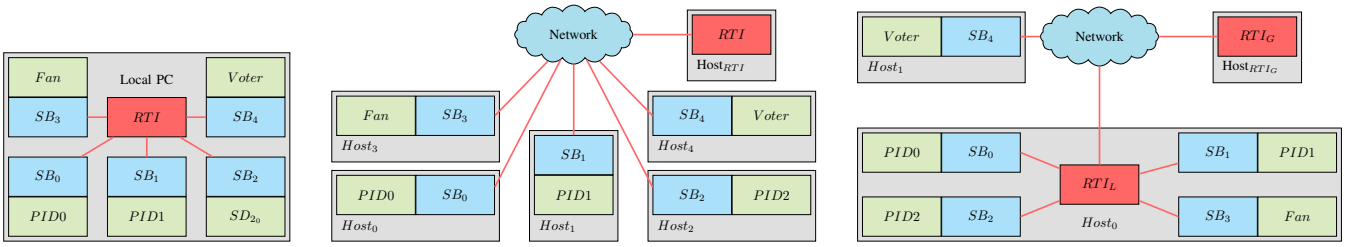
The state machine shown in Figure 3 represents the main loop of the co-simulation module. First, the module checks if a new step has to be performed which is signaled by the flag *stepPending*. If the RTI granted a time advance to a time before the requested one, there might be pending internal or additional, unknown events from other devices. Hence, the request needs to be repeated. Otherwise, the co-simulation module can request the next event's logical time from the event schedule. If all events are processed, the simulation is finished and the co-simulation module switches into the *stopped*-state. This case is signaled by returning a value called *maxEventTime* which is equal to the maximum 64-bit integer. Before the RTI sends the time advance grant, it transmits all messages destined to the federate. The co-simulation stores the data in the input packet buffer and notifies the delay-management module about the availability of new packets. Once the step is finished, all output packets are sent and the co-simulation module can request the next time advance.

C. Packet handling

Both, the state-estimation and the fault-injection modules are not in the focus of this paper. Hence, the delay-management module only triggers the fault-injection module providing pointers to the packets received. In future works, the fault-injection module will inject faults according to the EN 50159 standard into the packets. Since it shall be possible to observe the injection in the monitoring module even if packets are omitted, an additional buffer is considered. The fault-injection module will check if the packet shall be omitted or if another fault has to be injected. It inserts non-omitted packets into the buffer and forwards the related pointers to the wrapper module. Since the fault-injection is not implemented yet, all packets are forwarded without any fault.

Before the simulation starts, the wrapper module establishes the FMI connection to the simulated device by loading the specified FMU. At runtime, it accesses the packets related to the received pointers and forwards them via FMI. Here, we extended the standard by adding a new function *fmiSetPacket*. A packet is stored as char array. Instead of copying it like in *fmiSetString*, we just pass a pointer to the memory location. This way we decrease the pass-through time in the simulation bridge.

Once the FMU received all packets, it can perform a simulation step. The wrapper module calls *fmiPerformStep* which processes all events until the current granted time is reached. Thereby, all output packets are stored in a buffer. The return value of *fmiPerformStep* allows the wrapper module to determine the status of the FMU. If the status is *discard*, *error* or *fatal*, the wrapper modules signals the termination of the simulation execution. In case of *pending*, no output packet is received and the co-simulation module can advance in time. Otherwise, the status is ok and the wrapper module can request the number of output packets. The function *fmiGetString* was changed to enable the wrapper module to loop over the packets in the FMU-internal buffer. This way, it is not required to define FMI-variables for each possible output packet. Instead of copying the packets, only pointers are provided to the wrapper module. Each packet is analyzed by the wrapper to determine the related HLA interaction class and the packet's send time. While the interaction class is based on the destination MAC-address, the send time must be added explicitly. For this, we use an additional FMI interface variable. The wrapper module inserts all packets into the output packet buffer. Furthermore, it notifies the co-simulation module to send the packets and to advance in time.



(a) Local setup (b) Communication via LAN or the Internet (c) Example for hierarchical RTI

Fig. 4. Different topologies used during the evaluation. (a) All simulations and the RTI are executed on the local PC. (b) The RTI is moved to a remote server and the hosts communicate either via a LAN or the Internet (secured via VPN). (c) Topology to evaluate a hierarchical RTI. The voter communicates directly with the remote RTI (RTI_R) via a network (see (b)), the remaining simulations with a local RTI (RTI_L).

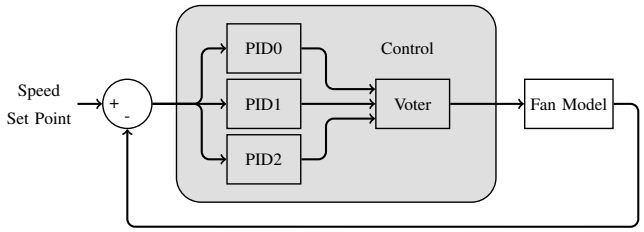


Fig. 5. Fan control with triple modular redundancy.

IV. Framework Instantiation

To perform an evaluation of our framework, we use a fault-tolerant fan control application as device under test. It is presented in Figure 5. A PID-controller receives the current speed of the fan and a reference speed from the user. It subtracts the current speed from the reference to calculate the difference which is the input for a PID control algorithm. Fault-tolerance is reached using a triple modular redundancy concept [22]. The PID controller is triplicated and connected to a voter which receives the PID outputs. This setup is able to detect a failure in one controller based on a majority election in the voter. If two controllers fail, all voter inputs differ. The voter then puts the system into a safe state by running the fan at its maximum speed. Otherwise, it forwards the control value to the fan model which calculates the fan's new speed. Each subsystem is connected to a simulation bridge via FMI.

The fan model is based on an Intel E97379-001 fan. It has a basic speed which is increased using a controllable portion. Both can be modeled as first-order delay elements using the following equations. Equation 1 represents the base speed while Equation 2 denotes the controllable portion. At runtime, the portions are added resulting in the final speed.

$$a(t) = 10 * \left(1 - e^{\frac{-t}{0.46}}\right) \; and \; G(s) = \frac{10}{1 + 0.46s} \quad (1)$$

$$a(t) = 10 * \left(1 - e^{\frac{-t}{0.87}}\right) \; and \; G(s) = \frac{10}{1 + 0.87s} \quad (2)$$

There are three topologies used as shown in Figure 4. In the first topology (a), all simulations, simulation bridges and the RTI are executed on the same PC. In the other two topologies we distributed the setup. In (b), the RTI runs on a remote PC which is connected via a LAN or the Internet. To protect confidential data, a VPN is required if the Internet is used. In (c), the RTI is distributed in a hierarchical manner. The fan model as well as the PID-controllers are connected to a local RTI instance. Similar to the voter, this RTI instance communicates with its remote, parent RTI via a network. In our evaluation in Section V, we compare the results of all topologies.

V. Evaluation of Test Results

During the evaluation, we selected eleven simulation durations for executing the fan use-case. The durations start with 1s and continue from 10s to 100s in 10s-steps. Each duration was executed 100 times in case of the local PC- and LAN-setups and 20 times in the other cases. In this section we analyze the simulation durations, the delays introduced by the simulation bridges as well as the influence of the communication network on the communication delay.

Figures 6 and 7 illustrate the simulation durations (placed on the y-axis) compared to the selected, simulated time which is shown on the x-axis. While Figure 6 gives a more detailed view on the durations of the LAN and the local setups, Figure 7 illustrates the Internet-cases. Since the communication via the Internet introduces large delays, the Internet-cases are shown in a scaled coordinate system.

The black dash-dotted graphs with filled square marks represent the case when all subsystems are executed on the local PC (the local setup), the gray ones the second case when the RTI is executed on a host in the same LAN (circle marks, densely dashed line) and the hierarchical LAN-setup (x-marks, loosely dashed line). While the Internet-case is depicted as dash-dotted black line with filled circle marks, a densely dashed line with filled triangle marks represents the hierarchical Internet-case. In each graph, the marks represent the average of all simulation durations measured for the related simulated times. Furthermore, we included a black, solid line. This line denotes the case when the simulation duration is equal to the simulated time, hence we call it *real-time*. If a graph is below this line, the simulation is executed faster than real-time, otherwise it is executed slower.

Each Figure shows a linear correlation between the simulated time and the simulation duration. For every topology we calculated a regression line which represents the linearity. The

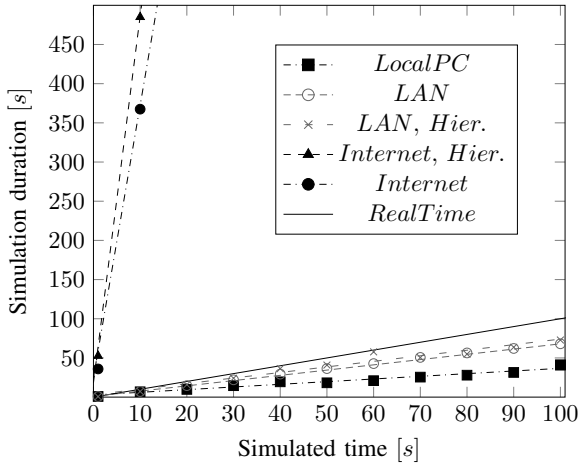


Fig. 6. Simulation durations for simulated times until 100s.

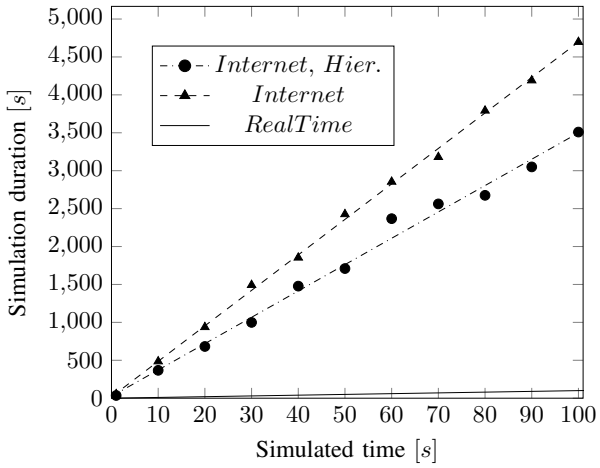


Fig. 7. Simulation durations for simulated times until 100s if a VPN is used.

regression slope for the local setup is $0.34t$, the one related to the communication to a host in a LAN is $0.67t$. The slope in the hierarchical LAN-setup accounts to $0.71t$. All three cases are executed faster than real-time (slope: $1t$). Pinging the local network interface results in round trip times of $0.05ms$ to $0.1ms$ whereas the LAN introduces delays around $0.5ms$. Using the Internet shows the influence of the network and the application of security mechanisms. In our case, round trip times of $15ms$ to $30ms$ were measured. Compared to pinging the local interface, the delays introduced here are 300 times larger. Therefore, also the simulation durations in the Internet-cases are much longer as shown in Figure 7 and in the regression slopes of $34.78t$ (hierarchical VPN setup) and $46.79t$ (all bridges connected directly). The results further show the benefit of a hierarchical RTI if the communication delays are large. In theory, there is less communication via the network due to the local RTI instance. However, the data exchange between the RTI instances results in a high overhead which limits the performance. In a LAN, the communication delays are short enough so that there is no benefit of a hierarchical solution. As soon as the delays increase, the synchronization overhead is mitigated and the simulation duration reduces.

The influence of the communication delay on the simulation duration is shown in Tables I to III. These delays are introduced by the heterogeneous network connecting the bridges.

Table I compares the interaction delays between the simulation bridges in ms. There are five interactions: *SensorRPM* which is a multicast from the fan to the three PID-controllers, three interactions PWM_i sent from the PID-controllers to the voter and interaction *PWM* which closes the loop between voter and fan. For each interaction and topology, the delay between the simulation bridges is depicted. The timestamps used for the calculation are taken in the communicating simulation bridges before sending and after receiving. As a consequence, the delays include the delays introduced by the network and also the time required for synchronizing the bridges using the HLA time management. This effect can be seen in case of the PWM_i-interactions. The PID-controllers calculate the control-value around the same time and send the interaction directly. However, the delay for PWM_0 is much smaller than the one for PWM_2. The reason is that the reception of the interactions by the voter is scheduled for logical times 4 (PWM_0), 5 (PWM_1) and 6 (PWM_2) while the values are computed at logical time 3. In between, the HLA time management has to synchronize the time advance of each federate.

Comparing the regression lines with the results in Table I, there is a similar relationship between the interaction delays and the simulation durations for each topology. The average fraction of the Internet delay divided by the delay of the LAN is 62.111. Using the delay of the local setup as divisor, the fractions are 1.911 (LAN), 2.080 (LAN, hierarchy), 35.964 (Internet, hierarchy) and 118.929 (Internet). The related fractions between the regression slopes are 69.836 (Internet / LAN), 1.971 (LAN / Local), 2.088 (LAN, hierarchy / Local), $102,29$ (Internet, hierarchy / Local) and 137.618 (Internet / Local). Even if the fractions are not exactly the same as the regression slopes, the dimensions show the main influence of the communication. This assumption is further proved by the analysis of Tables II and III as shown in the following.

The average pass-through times of the simulation bridges for sending and receiving packets are shown in Table II. Similar to Table I, the times are depicted for each topology. Since the packets have to traverse more modules on the incoming side (see Section III), the pass-through times are longer than the ones for sending. While the times for passing the bridges in the local setup and the LAN case are quite similar, the VPN has a strong influence. Using a VPN, all traffic is secured and routed to the connected VPN-server, not only traffic related to the simulation execution. Applying the security algorithms requires processing power which affects the execution times of the bridges. However, the order of magnitude of the pass-through times is still much smaller than the interaction delays. The decreasing times for longer simulated times can be constituted with a smaller effect of

TABLE I
INTERACTION DELAYS BETWEEN THE SIMULATION BRIDGES IN THE DIFFERENT USE-CASES.

Interaction	SensorRPM Fan - PID0	SensorRPM Fan - PID1	SensorRPM Fan - PID2	PWM0 PID0 - Voter	PWM1 PID1 - Voter	PWM2 PID2 - Voter	PWM Voter - Fan
Delay Local PC $[ms]$	0.395	0.387	0.387	0.527	0.832	1.219	0.630
Delay LAN $[ms]$	0.787	0.780	0.777	0.877	1.466	2.231	1.324
Delay Hier. LAN $[ms]$	0.867	0.923	0.907	1.109	1.557	2.443	1.041
Dela Hier. VPN $[ms]$	11.631	11.577	11.609	19.231	31.895	51.926	28.327
Delay VPN $[ms]$	47.892	47.224	46.641	48.838	94.327	143.990	91.065

TABLE II
AVERAGE PASS-THROUGH TIME OF THE SIMULATION BRIDGES.

Simulated time	Receive $[\mu s]$					Send $[\mu s]$				
	Local PC	LAN	Hier. LAN	Hier. VPN	VPN	Local PC	LAN	Hier. LAN	Hier. VPN	VPN
1	67.848	66.199	56.711	129.960	164.236	61.410	53.160	49.118	119.318	146.618
10	63.189	57.075	53.810	173.459	147.863	54.515	44.967	44.514	146.233	125.755
20	61.813	61.554	55.204	79.968	161.994	47.747	47.739	45.779	73.849	135.102
30	62.076	61.712	54.002	80.723	145.340	47.450	47.441	45.207	74.455	120.380
40	61.422	61.655	57.185	80.238	129.933	46.803	47.092	49.192	73.711	107.733
50	50.216	54.760	54.001	81.562	149.553	39.498	41.931	45.371	74.524	124.002
60	48.429	56.431	58.601	81.740	130.794	38.261	42.912	50.877	74.426	107.480
70	49.701	55.343	57.080	195.886	149.933	38.939	42.038	45.651	161.078	123.126
80	48.458	54.598	56.671	80.954	150.600	38.150	41.215	44.923	73.845	123.033
90	48.489	52.858	55.603	82.375	134.215	38.224	40.339	44.998	74.041	109.842
100	47.698	53.420	57.244	82.040	150.495	38.757	40.651	46.336	74.400	122.018

TABLE III
FRACTIONS OF SIMULATION BRIDGE DELAYS COMPARED TO SIMULATION DURATION

Simulated time		1	10	20	30	40	50	60	70	80	90	100
Local PC	Delay $[s]$	0.079	0.682	1.333	2.022	2.661	3.065	3.186	3.828	4.258	4.835	5.377
	Duration $[s]$	0.826	6.828	9.842	14.588	19.522	18.280	20.987	25.520	27.996	31.499	41.030
	Fraction	0.096	0.100	0.135	0.139	0.136	0.168	0.152	0.152	0.135	0.153	0.131
LAN	Delay $[s]$	0.075	0.631	1.343	1.999	2.684	3.008	3.698	4.113	4.739	4.878	5.877
	Duration $[s]$	0.788	7.155	14.741	21.906	29.659	33.653	41.248	47.469	55.736	62.550	66.706
	Fraction	0.095	0.088	0.091	0.091	0.091	0.089	0.090	0.087	0.085	0.078	0.088
Hier. LAN	Delay $[s]$	0.064	0.599	1.231	1.812	2.585	3.024	3.988	4.395	4.971	5.528	6.324
	Duration $[s]$	0.680	7.223	15.919	24.072	36.358	40.919	57.748	50.434	54.885	63.634	72.872
	Fraction	0.094	0.083	0.077	0.075	0.071	0.074	0.069	0.087	0.091	0.087	0.087
Hier. VPN	Delay $[s]$	0.153	1.945	1.858	2.812	3.721	4.718	5.666	15.236	7.487	8.521	9.463
	Duration $[s]$	35.650	356.987	744.058	1030.94	1309.71	1608.41	1967.28	2684.17	2892.25	2882.68	3528.94
	Fraction	0.0043	0.0040	0.0025	0.0027	0.0028	0.0029	0.0029	0.0077	0.0026	0.0030	0.0027
VPN	Delay $[s]$	0.188	1.664	3.619	4.858	5.793	8.334	8.718	11.656	13.355	13.398	16.636
	Duration $[s]$	52.633	484.952	936.809	1492.54	1853.26	2426.03	2853.82	3179.66	3791.84	4190.04	4697.01
	Fraction	0.0036	0.0034	0.0039	0.0033	0.0031	0.0034	0.0031	0.0037	0.0035	0.0032	0.0035

outliers due to more inputs. While analyzing the monitoring files, we found a few number of pass-through times in the dimension of ms instead of μs.

To analyze the general influence of the simulation bridge pass-through times on the simulation durations, we calculated the delay sum and divided it by the duration. The resulting fractions are shown in Table III. The pass-through times in the local setup account for 13.6% (in average) of the overall duration while this fraction is reduced to 8.8% (LAN), 8.3% (LAN, hierarchy) and 0.3% (both Internet-cases), respectively. Together with the previous results, these fractions elucidate the strong influence of the communication delays on the simulation duration.

VI. CONCLUSION

Distributed SIL simulations via the Internet or LANs is a promising approach to simplify the integration and testing of complex, networked embedded systems. However, the current state-of-the-art shows a lack of such simulation tools focusing on a network-centric co-simulation. The presented framework provides such mechanisms and it allows to validate the communication behavior of networked systems already during

early development steps. Due to the framework's distributed character, it is possible to validate the software algorithms directly at the manufacturer's sites instead of shipping them to a central place.

We implemented the framework based on a combination of the HLA and FMI co-simulation standards. The evaluation using a fault-tolerant fan control application shows a linear dependency between the simulated time and the time required to execute the simulation. However, the simulation is highly affected by the communication network used to connect the simulation bridges and the delays it introduces. As long as the device under test has similar dimensions like the fan with five communicating subsystems and communication periods of 10ms, it is possible to run simulations faster than real-time or with real-time requirements. However, this is not possible if there is a remote component which requires the Internet to communicate with the remaining DUT. In this case, we can only run setups in which the system's time can be paused, so there are no real-time requirements. Furthermore, we did not analyze the temporal characteristics for large scale DUTs. Such an analysis as well as handling the communication delays will be topics for future works.

ACKNOWLEDGMENT

This project has received funding from the European Unions 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 Proceedings of International Conference on Computing, Networking and Communications (ICNC), 2013 , pp. 315-320. IEEE, 2013.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] A. Cimatti, R.Raffaele, A. Lazzaro, I. Narasamdya, T. Rizzo, M. Roveri, A. Sanseviero, A. Tchaltsev. "Formal Verification and Validation of ERTMS Industrial Railway Train Spacing System". In Computer Aided Verification, pp. 378-393. Springer Berlin Heidelberg, 2012.

[8] F. Corbier. "Accelerate the Development of Certified Software for Train Control & Monitoring Systems". In 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016). Toulouse, France, 2016.

[9] 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.

[10] 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.

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

[12] 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.

[13] M. Ficco, G. Avolio, F. Palmieri, A. Castiglione. "An HLA-based framework for simulation of large-scale critical systems". Concurrency and Computation: Practice and Experience, Vol. 28, pp. 400-419, 2015.

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

[15] H. R. Hertzfeld, A. N. Link, N. S. Vonortas. "Intellectual property protection mechanisms in research partnerships". Research Policies, Special Issue on "Property and the pursuit of knowledge: IPR issues affecting scientific research", pp. 825-838, 2006.

[16] 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.

[17] "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification". IEEE Std 1516.1-2010 (Revision of IEEE Std 1516.1-2000), 2010.

[18] 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.

[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] Klein, U., S. Straburger. "Zivile Anwendungspotentiale der High Level Architecture (HLA)". Symposium Neue Technologien in der wehrtechnischen Simulation, Mannheim, 1997.

[21] H.-Y. Kim, N.-J. Lee, D.-C. Lee and C.-G. Kang. "Hardware-in-the-Loop Simulation for a Wheel Slide Protection System of a Railway Train". In IFAC Proceedings Volumes of the 19th IFAC World Congress, pp. 12134 - 12139, 2014.

[22] H. Kopetz. "Real-Time Systems: Design Principles for Distributed Embedded Applications". 2nd Edition. Springer Publishing Company, Incorporated, 2011.

[23] 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.

[24] T. Pieper, R. Obermaisser. "Distributed co-simulation for software-in-the-loop testing of networked railway systems". In Proceedings of the 7th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, pp. 24-28, 2018.

[25] 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.

[26] 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.

[27] 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.

[28] 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.