D3.1 – Report on State-of-the-Art Analysis and Initial Requirements for the Distributed Simulation Framework

Project number:	730830
Project acronym:	Safe4RAIL
Project title:	Safe4RAIL: SAFE architecture for Robust distributed Application Integration in roLling stock
Start date of the project:	1st of October, 2016
Duration:	24 months
Programme:	H2020-S2RJU-OC-2016-01-2

Deliverable type:	Report (R)
Deliverable reference number:	ICT-730830 / D3.1 / 1.2
Work package	WP 3
Due date:	December 2016 – M03
Actual submission date:	30th of December, 2016

Responsible organisation:	University of Siegen
Editor:	Tobias Pieper
Dissemination level:	Public
Revision:	1.3

Abstract:	Describes the SOTA of distributed simulation frameworks including existing solutions from other domains. In addition, the deliverable contains the initial requirements for the concept of the S4R simulation and validation framework.
Keywords:	Distributed co-simulation, HIL, SIL, T2G

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

Editor

University of Siegen, SIE

Contributors/Reviewer (ordered according to beneficiary numbers)

TTTech Computertechnik AG, TTT

IKERLAN, S. Coop, IKL

UniControls A.S., UNI

Technikon Forschungs- und Planungsgesellschaft mbH, TEC

TÜV SÜD Rail GmbH, TÜV

IAV GmbH Ingenieurgesellschaft Auto und Verkehr, IAV

IFSTTAR – Institut Francais des Sciences et Technologies des Transport, de l'Amenagement et des Reseaux, IFS

Executive Summary

Integrating and testing railway components are crucial steps in the development progress. Hence, one main objective of the ongoing European research project Safe4RAIL lies in the concept design and the proof-of-concept implementation of a distributed simulation and validation framework. It aims on a network-centric simulation at system level providing a time-accurate simulation of in-train communication networks with co-simulated end-systems. Those end-systems are securely coupled using public communication networks. Using the framework can radically improve the integration and testing process compared to today's practice.

In this deliverable, the state-of-the-art in distributed simulation frameworks and other solutions of interest is presented. Such solutions refer to software- and hardware-in-the-loop simulation and the simulation of wireless railway networks. Furthermore, different approaches for the evaluation of the framework are presented as well as test automation and fault-injection techniques. For each topic, the deliverable defines design goals for following tasks in the project and it points out the research gap to the state-of-the-art. The design goals have to be finalized to requirements in the following months.

There are already various approaches available that can be utilized during the SAFE4RAIL project. The combination of the Functional Mock-up Interface (FMI) and High Level Architecture (HLA) standards is a promising approach for the co-simulation of different simulation tools since FMI is already widely used and supports various simulation tools. Furthermore, there are solutions for the inclusion of Software- and Hardware-In-The-Loop (SIL/HIL) or the injection of faults into the simulation. HIL and SIL simulation are used to test the communication between mobile communication gateways and peers in the train-to-ground communication.

However, there are major research gaps the existing solutions do not solve yet. For example, the interconnection of simulation tools introduces delays which are inapplicable in providing real-time behavior for HIL simulation. The universal applicability of the plant models for user interaction during tests is another high requirement. Further requirements are related to timing, configuration, applicability and safety in the co-simulation framework. Those gaps and the definition of the final requirements have to be solved during the next months.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

Integration of railway components and their testing are crucial development steps. One objective of the ongoing European research project Safe4RAIL lies in the concept design and a proof-of-concept implementation of a distributed simulation and validation framework. Using this framework, the integration and testing process can be radically improved compared to the today's practice.

The distributed simulation and validation framework will support Software- and Hardware-In-The-Loop (SIL/HIL) testing as well as the secure coupling of simulators and physical systems via public communication networks. The goal is a network-centric simulation at system level providing a time-accurate simulation of in-train communication networks with co-simulated end-systems. As a consequence, train manufacturers and suppliers will be able to perform an early validation before a complete train-system is available.

This deliverable presents a State-of-the-Art (SotA) analysis related to the topics of the distributed simulation framework and other solutions of interest. For each topic, it defines design goals for the following tasks in the work package and outlines the research gap to the state-of-the-art. In the following months, those design goals have to be finalized to requirements for the distributed simulation framework. The structure of the deliverable is organized as follows:

- Chapter 2 deals with available SIL and HIL frameworks from various domains like railway, automotive or distributed SIL/HIL testing. It defines standard protocols and interfaces and general requirements for HIL testing as well as timing requirements for both techniques.
- Chapter 3 gives an overview on Train-to-Ground (T2G) wireless communication networks and defines requirements for the Quality of Service testing and the test environment of the Train Control and Management System (TCMS) T2G interface.
- Chapter 4 continues with distributed simulation frameworks and co-simulation. Additionally, it analyzes the state-of-the-art of security mechanisms for the communication via public communication networks. The requirements defined in this section concentrate on the (distributed) co-simulation, timing, applicability, configuration as well as security.
- Chapter 5 deals with the state-of-the-art of safety evaluation concepts. It defines requirements for certification and the support of a safety-case.
- Chapter 6 presents tools for test operation and test automation and defines the related requirements to that topic.
- Chapter 7 finishes with an overview on existing fault-injection mechanisms, tools and requirements to evaluate extra-functional properties.

Chapter 2 SIL and HIL frameworks

2.1 Test Cases

The main usage of testing with SIL and HIL is supporting system integration and verification of the implemented features.

The following paragraphs will present typical test cases for SIL and HIL simulation. Their explanation will facilitate the understanding of the impact the HIL Simulation Framework (HILSF) will have. These test cases are not TCMS specific but common in industrial development of complex software controlled systems.

2.1.1 System Level

The detailed discussion of the different test cases requires a short definition of decomposition of the TCMS.

- TCMS groups all control devices controlling the physics of a rail vehicle.
- The term system in this context refers to the delivery item of a supplier of the rail vehicle OEM (Original Equipment Manufacturer). The delivery item consists of a controller – by itself part of the TCMS – and its controlled part of the physics of the rail vehicle, e.g. brakes or propulsion.

For easier understanding the following artificial, an abstract system can be used. It shows a Controller containing two functions for the control of rail vehicle propulsion system.

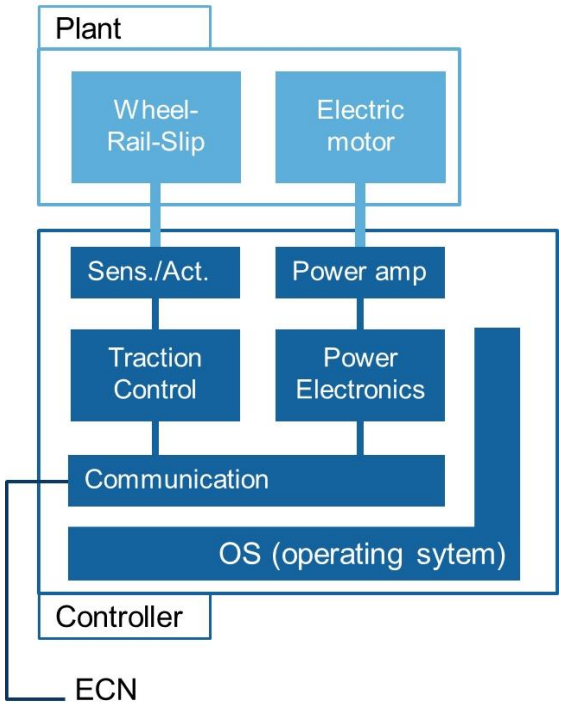

Figure 2.1 TCMS controller example

2.1.2 Function development – using MIL / SIL

The focus in the function development using Model-In-The-Loop (MIL) and SIL lies in the validation of the algorithm for controlling a subset of the plant physics. Challenges in the functional development test case are related to modelling and simulation.

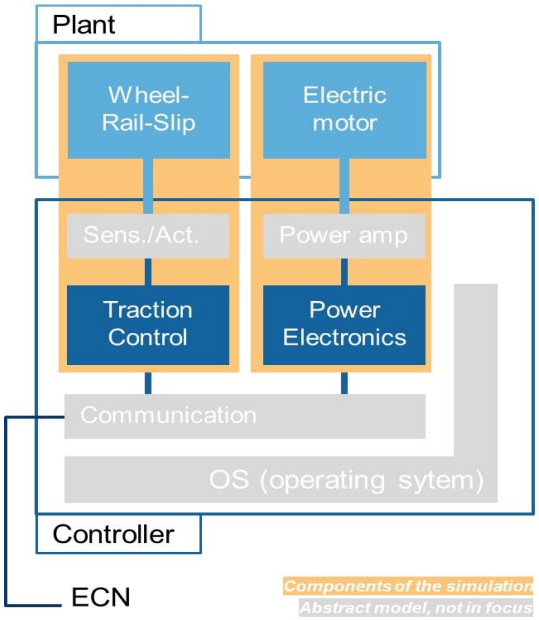

Figure 2.2 Function development setup

The model includes an exact representation of the plant behavior to be controlled and each subsystem shows different timing characteristics (e.g. ranging from 1ms for slip control to 1µs for electric field control).

In this test case, a multi-tool-simulation is required to connect, for example, the function development tool with an appropriate modeling tool for the plant physics. Furthermore, simulations are coupled (i.e. each model is a timed model) and the simulation is not executed in real-time. Instead of using event-based simulation, it is signal-based.

2.1.3 Plant driver development – using MIL / SIL / HIL

This test case focuses on the development of driver / sensor interfaces appropriately interfacing the plant. In contrast to the previous case, components are developed independently from each other as they do not interfere (most often Commercial Off-The-Shelf (COTS) components).

The model exactly represents the plant behavior and the hardware interface of the controller. This interface is implemented using specialist tools without a relation to the tool chain for implementing the controller software.

Similar to the development using MIL and SIL, the simulation is not real-time and signal-based. Furthermore, simulations are also coupled, i.e. each model is a timed model.

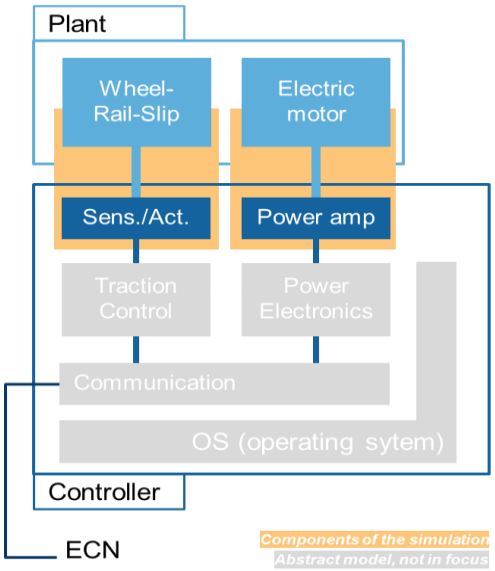

Figure 2.3 Function test simulation setup

2.1.4 Software integration test

The software integration test is performed after the algorithms as software modules onto the OS with its timing and task-architecture. It checks if the Operating System (OS) configuration changes the module's behavior. The modules can be developed independently and by distributed suppliers. Typically, the integration is a centralized activity of the OEM.

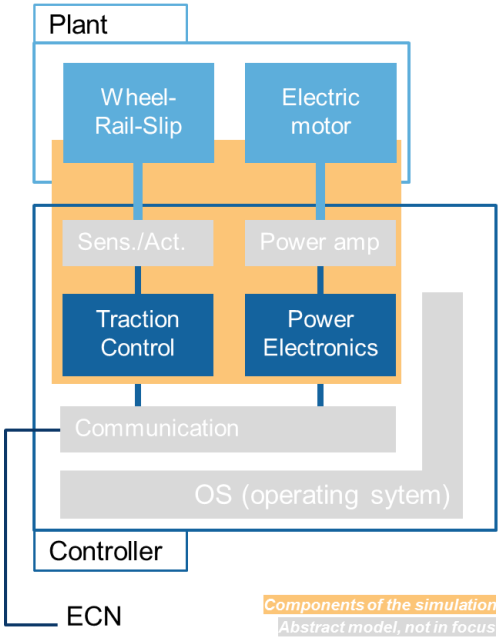

Figure 2.4 ECU (Electronic Control Unit) Software integration simulation setup

Challenges with respect to modelling are a series function implementation of the controller and not the model. In addition, a broad spectrum of plant models is required to be integrated. Each module loses its own timing due to the series function. Only the operating system clock remains and all modules are executed as run to completion tasks. Due to the very different nature of the plant models in terms of timing and / or modelling approach, only an abstract model is used.

If an OS emulation is available, the usage of SIL is possible. Otherwise, HIL is used. Similar to the other cases, the simulation is signal-based. The timing characteristics suggest not to distribute the execution of the modules.

2.1.5 Communication (COM) layer test

This test performs a HIL execution of a full validation test sequence for the Ethernet Consist Network (ECN) protocol implementation. The execution is performed separately for each controller and typically by the hardware supplier but not the railway manufacturer. The according reference is the ECN standard, therefore no cooperation is needed.

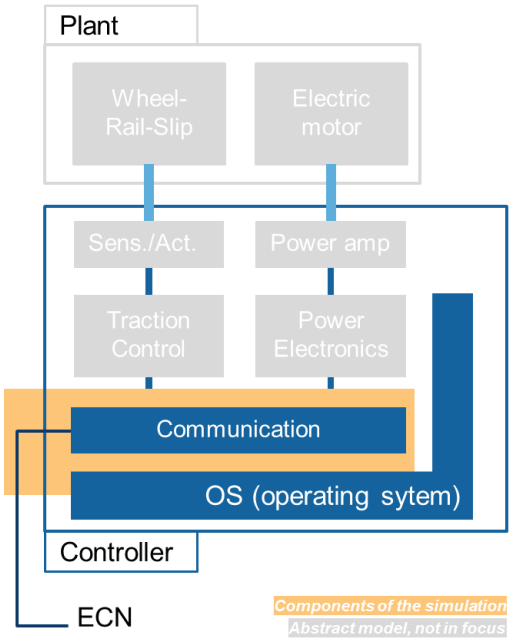

Figure 2.5 COM layer test simulation

The model contains a complete implementation of a certified test sequence for the ECN protocol. In contrast to the previous cases, an event-based simulation is possible. The simulation is consistent and automatically creates reports for certification purposes.

2.1.6 COM configuration test

The COM configuration test validates the correct signal routing and communication supervision of a vehicle-specific ECN configuration. It is executed by the OEM responsible for the correctness of the ECN network and must be typically performed for each individual ECN configuration.

In this case, no plant models are required. Moreover, standard-test sequences are instantiated to test the specific ECN configuration of the vehicle. Again, event-based simulation is possible and HIL is used established by the producer.

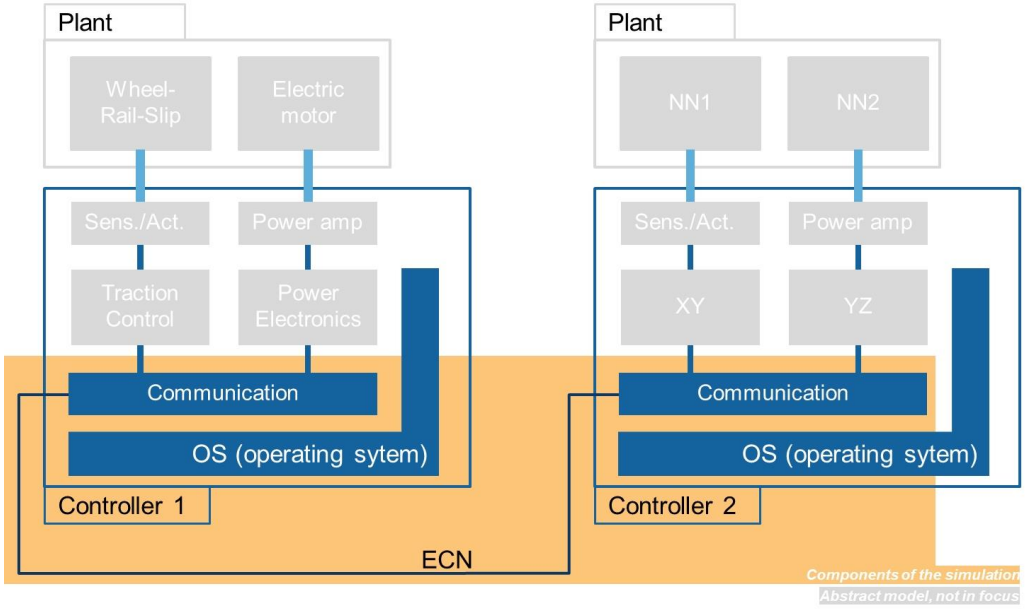

Figure 2.6 COM configuration test setup

2.1.7 HIL - ECU test

The focus in this test lies on the validation of the integrated controller functions in the final hardware. The vehicle manufacturer thereby completely validates the requirements implementation for all functions using the integrated controller.

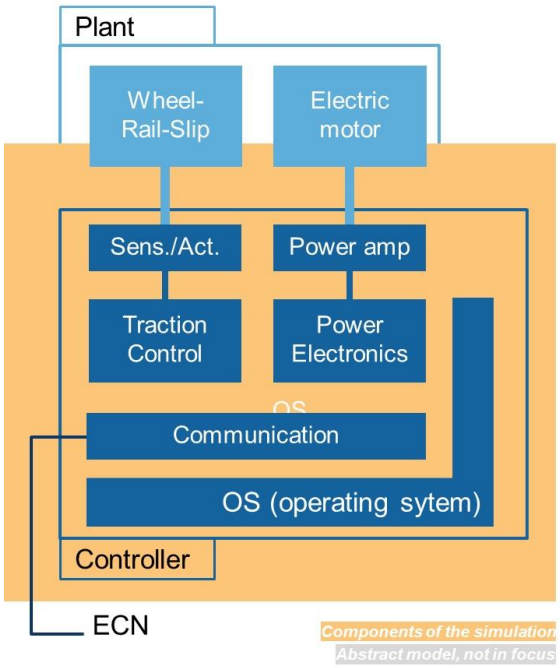

Figure 2.7 HIL ECU test setup

Test sequences are derived from abstract requirements and must be detailed to level that they stimulate the series interfaces of the controller. The set of test cases is fixed, thus plant models are built to purpose as follows. Testing the functions requires an exact representation of the corresponding plant behavior. As the different parts of the plant impose a broad spectrum of properties (e.g. simulation step size), typically not one universal plant model is used. Instead, each control function group and plant property is a specialized model.

In parallel to the test execution, the test framework must change and instantiate the plant models addressed by each test sequence. Due to the different plant models, debugging of problems imposes difficulties, as the same problem must be represented in different test sequences. This test uses a signal-based simulation.

2.1.8 System function exploration

Investigating the behavior of a distributed function across multiple controllers is performed in the system function exploration. For example, a subset of the vehicle functions across all controllers (e.g. diagnostic function or the driving behavior) is tested.

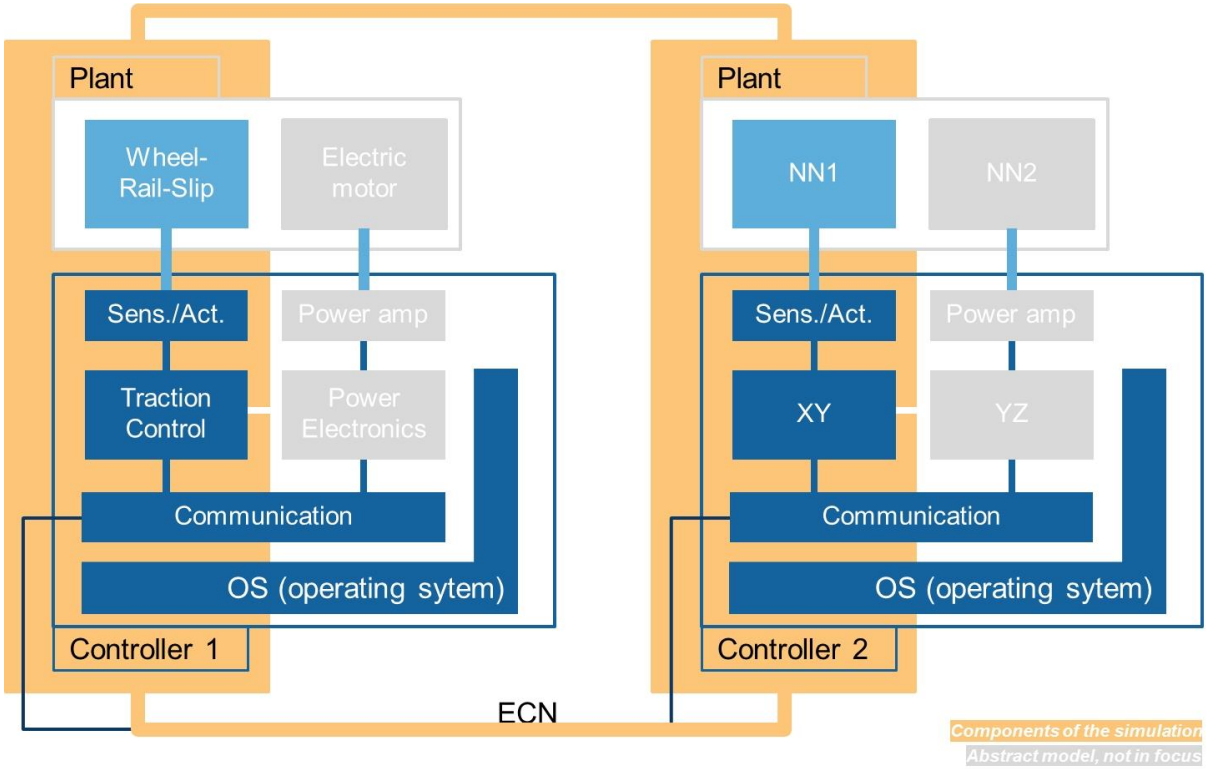

Figure 2.8 System function exploration setup

Similar and closely coupled functions are implemented in parallel by different teams or companies. As they implement the same requirements, an early check of the integrated behavior is mandatory. Further, interoperability of controllers of different vendors can be assessed only in such integrated setup across controllers. The term “exploration” addresses the fact that often no standardized test sequences are executed. The tests focus on debugging claims from the vehicle operation and the behavior’s exploration of the functions in the context of changed boundary conditions, e.g. a changed operation procedure of the vehicle.

The plant models exactly represent the chosen aspect of the vehicle function and their implementation across the different controllers. Therefore, the different plant models must be suitable for integration. The simulation is executed in SIL or HIL setups. A distributed execution is promising since it reduces the need for unifying the modelling tools and the modelling method across companies. The timing requirements on the connection of the distributed plant models are often more severe than the connection of the controllers. Those are stiff coupling for the plant and loose coupling for the controller by ECN. Due to the plant models, a signal-based simulation is performed.

2.2 Design goals for SIL and HIL frameworks

The design goals defined for SIL and HIL frameworks are related to four different topics. In Section 2.2.1, general assumptions are explained followed by standardized interfaces and protocols for HIL. In the next sections, more design goals based on proprietary solutions for OEMs follow in Section 2.2.3 and general design goals for simulation frameworks follow in Section 2.2.4. At last, design goals related to timing in HIL and SIL frameworks are presented (see Section 2.2.5).

2.2.1 General assumptions

First, we should also take into consideration properties of HIL distributed simulation frameworks. The reason is that some requirements are common.

Most of the individual mechanical, pneumatic and electric devices which belong to the basic technology of railway cars, relevant sensors, converters, actuators and also processor based and configurable devices with simply definable functionality can be considered as Systems Under Test (SUTs) for which the usage of HIL frameworks may be meaningful. However, requirements arising relative such devices should be some subsets of requirements for HIL frameworks in their relationship with integrated TCMSs. This includes their programmable units with complex functionality.

The mentioned devices with relatively simple functionality are usually tested by means of test benches with relatively simple and long-time changeless behavior. Simulation of complex environment of these SUTs is mostly not necessary. A few and fixed sequences of stimuli are usually sufficient for verifying the correct behavior of the considered SUTs.

In contrast to the above mentioned “simple devices”, the core programmable units of TCMSs are tested by using simulation models of the whole environment. Test cases can be related to the units’ functionality depending on their firmware, application software, communication means, configuration data and complete parameterization. In this context, the models of the whole environment include the sub-models of the “simple devices” and attained degree of fidelity must be evaluated carefully in individual cases. Sometimes this degree is tuned or even optimized with respect to compromises between model fidelity and demanded computing power. The Simulation fidelity metric is a related, important topic to be considered regarding the reproducibility of tests.

The following technical challenges regarding non-determinism of the SUT and HILSF shall be considered influencing the validity of the simulation:

a) Execution of tasks of simulation programs on the side of HIL simulation framework is not explicitly synchronized with tasks of programmable units of SUTs. It means that SUT do not need to provide any synchronization signals for the simulation framework.

b) If a SUT includes two or more processor units with asynchronous execution of their tasks, then its reactions to exogenous stimuli are generally not deterministic. Any test environment cannot ensure strict repeatability of tests for such SUTs in terms of the same way of internal path of execution in the SUT. It is necessary to prescribe some metrics as well as its limit values as fidelity criteria for individual tests. By this, it is possible to implement relevant evaluation software and to achieve some kind of weak repeatability of tests.

c) Regardless of non-deterministic behavior resulting from unsynchronized processes on the side of SUTs, a further source of problems arises. This problem is related to the repeatability of HIL simulation runs and tests in the application of analog I/O channels. Even if all timing problems can be eliminated in some cases, real Digital/Analog (D/A) and Analog/Digital (A/D) converters always bring non-deterministic behavior into channels. There are special cases when D/A converters of the SUTs and bound A/D converters of the simulation system cause essential repeatability problem. Then,

additional software level quantizers embedded after A/D converters may be useful. Unfortunately, it is not possible for the opposite signal flow to request such modifications of the SUTs' application software due to the fulfillment of the repeatability requirement.

As the general approach to solve the repeatability problem in HIL the definition of a simulation fidelity metrics and desirability of adequate supporting tools available in the intended simulation framework are needed.

2.2.2 Standard interfaces and protocols for HIL

In this chapter, standard interfaces and protocols for HIL are presented. Those interfaces and protocols are related to power supply and the communication with other components.

Power supply – directly powered devices
The HILSF shall provide the power supply for powering SUT devices according to the EN 50155 standard. It shall provide all the nominal voltage ranges specified in the standard, i.e. 24, 48, 72, 96 and 110 V DC.

Power supply – PoE devices
The HILSF shall provide means of powering a devices over Ethernet according to the IEEE 802.3af standard.

Ethernet connectivity – physical interface
The HILSF shall provide an Ethernet 100 BASE-TX interface with M12 connectors and D coding as well as an Ethernet 1000 BASE-TX interface.

Note

1000 BASE-TX is not yet standardized for the Train Communication Network (TCN) in the IEC 61375 standard family, but it is widely used in the industry and it is expected to be standardized soon.

Generic Internet protocol suite communication capability
The HILSF shall provide communication capability according to the subset of the Internet protocol suite including at-least the File Transfer Protocol (FTP), Hypertext Transfer Protocol (HTTP), Secure Shell (SSH) application layer protocols, Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Internet Control Message Protocol (ICMP), Internet Group Management Protocol (IGMP) transport layer protocols and the Internet Protocol (IP).

TCN protocol suite communication capability

The HILSF shall provide all communication protocols TRDP and Train Topology Discovery Protocol (TTDP) specified in the IEC 61375-2-3 and the IEC 61375-2-5 standards.

Standard TCN data classes support
The HILSF shall provide the capability of simulating following data classes for the communication over ECN and Ethernet Train Backbone (ETB) according to IEC 61375-1 standard. Those classes are supervisory, process, message, stream and best effort data.

FMI interface for HIL simulation adapter

The HILSF shall – by means of its HIL Simulation Adapter (see Figure 2.9) – provide an FMI interface for co-simulation of HIL / SUT(s) models connected to a simulator by communication channel(s) and/or by Input/Output (I/O) signals.

Optional extensions:

1. The model description file (modelDescription.xml according to the Functional Mock-up Interface (FMI), version 2.0) or its fundamental parts may be generated by an Integrated Development Environment (IDE) applied for development of the SUT, e.g. TCMS. Such an IDE on the one hand saves and processes configuration parameters regarding simulated devices. On the other hand, it handles individual messages or

datasets and their variables relative of data exchange between SUT and the simulated devices (mediating for the SUT's behavior of plants).

2. Some configuration parameters regarding the communication between HIL / SUT and the simulator are necessary for a functioning HIL Simulation Adapter. They could be defined by "Annotations" of "ScalarVariable" elements. Those in turn are elements of "ModelVariables" in several "fmiModelDescription" files. The configuration parameters can be further defined by HIL Simulation Adapter tool specific "VendorAnnotations" elements. In addition, some adoptions and designations have to be performed. Those are related to the definition of the Extensible Markup Language (XML) data structure of the specific "VendorAnnotation" and the reference rules from "Annotations" of individual variables to shared elements of the given "VendorAnnotation".

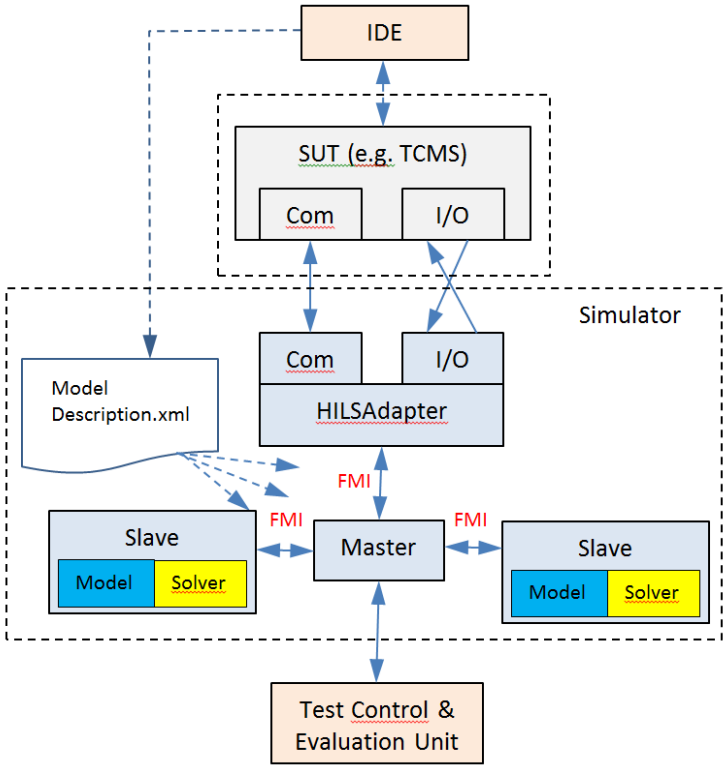

Figure 2.9 - FMI interface for HIL simulation adapter

2.2.3 More interface design goals for HIL in based on common proprietary solutions of OEMs

Design goals defined in this chapter are based on experiences with testing of the typical TCMS hardware.

Analog signal coupling conformable to industry standards
The HILSF shall provide modular means for configurable bidirectional coupling with SUT at least by following standardized electrical signals: current 0-20mA, +-50mA, Direct Current (DC) voltage +-10V and rheostat inputs in range 0 to 50kOhm. The resolution of the HILSF output shall be better than 10bits and the accuracy shall be better than 1%. Furthermore, the resolution of the HILSF inputs shall be better than 15bits while the accuracy shall be better than 0.5%. The data rate for both HILSF inputs and HISLF outputs shall be faster than 1ms.

Binary signal coupling conformable to industry standards
The HILSF shall provide modular means for configurable coupling with SUT at least as dry contact inputs, low power relay outputs, pulse counter inputs, DC voltage inputs and outputs.

The HILSF digital inputs and outputs as well as the dry contacts shall be able to work for nominal power supply voltages of 24 V, 28V, 36V, 48 V, 72 V, 96 V and 110 V DC.

On the one hand, the HILSF digital inputs shall be able to read the state of both SUT low and SUT high side outputs. In addition, the minimal current consumption of the HILSF digital inputs shall be equivalent to 200mW at the nominal power supply voltage. On the other hand, the HILSF digital outputs shall be able to drive both SUT inputs connected to the ground and SUT inputs connected to a battery voltage. The HILSF digital outputs shall be tolerant to short circuit and the nominal current shall be higher than 0.2A.

The propagation delay of HILSF digital inputs and output shall be lower than 1ms.

Driving the SUT pulse counter inputs or the SUT inputs for revolving sensors is realized by the HILSF outputs. Those shall be able to generate signals up to 30 kHz with rectangular or sinusoidal waveform and up to 4 output waveform with a phase shift of 90 deg. The amplitude shall be +-10V for the sinusoidal waveform and 0 to 12 or 0 to 24V for a rectangular waveform. To enable the simulation to revolve sensor malfunction, the HILSF shall be able to drive each output separately.

One advantage is the galvanic separation of digital inputs and outputs. It allows better coverage of real needs.

Individual parameterization of input channels
The HILSF shall allow the setup of filtration parameters determining cutoff frequencies of individual input channels. This includes the possibility to choose these parameters in accordance with sampling periods and related task periods.

Notes

This requirement coheres with the needful lowering of the phase delay enabled by shortening of sampling periods. This is desired in order to suppress delay scattering caused by asynchronous sampling and data processing. HILSF must sometimes simulate continuous systems and provide their pseudo-continuous outputs to be sampled by SUT devices asynchronously against HILSF tasks. The sampling periods may be roughly 10ms (for preprocessing) in the context of TCMS. However, the periods of generating relevant HILSF outputs has to be substantially shorter.

Configuration and parametrization
The HILSF shall be able to store multiple sets of the configuration parameters for each test case. It shall be possible to inject the parameters to the HIL by means of standard (e.g. Simple Network Management Protocol (SMNP), ONVIF (Open Network Video Interface Forum)) or proprietary configuration protocols. If the automated configuration of the HIL is not possible, it shall be able to display configuration instructions to the tester.

Test and measurement API
The HILSF shall provide an Application Programming Interface (API) for external measurements and evaluation tools as it provides a neutral API to the simulation components. During the simulation, it must allow co-execution with external tools.

2.2.4 General design goals for simulation framework

Design goals in this chapter specify general features for the test environment and the simulation framework used in TCMS testing. They are not necessarily limited to the HIL itself.

Sub-models and tasks
The HILSF shall allow defining simulation models composed of sub-models which should be assigned to several tasks with individual activation rules. This enables deterministic scheduling or concatenation.

Sub-models and programming languages

The HILSF shall provide all necessary tools for the development of simulation models by means of a graphic language.

Notes

This requirement is based on experience with simulations practiced by UNI Company, where PLC (Programmable Logic Controller) graphic languages Function Block Diagram (FBD) and Sequential Function Chart (SFC, according to IEC 61131-3) were applied.

Support of simulation model verification
The HILSF shall provide means supporting a user-definable verification process of individual sub-models. For those means, sets of measured test data can be obtained as time series of samples of all inputs and outputs of the sub-models to be verified.
Individual sub-model has to be verified autonomously for several test data corresponding with various modes and tests. Real-time as well as accelerated simulation shall be optional. Furthermore, simultaneous evaluation of several criterion functions shall be possible for tens of signals.

Predefined criterion function
The HILSF shall provide tools for evaluating wide-spread criterion functions of time series of deviations regardless their origination.

Statistical tools
The HILSF shall provide tools for the evaluation of stochastic processes.

Notes
Support of graphical outputs (histograms, correlation functions, power spectral densities) is grateful.

Pseudo-random generators
The HILSF shall provide sets of statistically independent and synchronizable pseudo-random white noise generators, optionally with Gaussian or uniform distribution. These generators shall be individually applicable as function blocks of simulation sub-models.

Noise coloring filters
The HILSF shall provide predefined objects of noise coloring filters applicable in connection with pseudo-random white noise generators. These filters should be parameterizable either in the time or the frequency domain.

Communication and feedback latency evaluation
The HILSF shall provide means facilitating statistic timing evaluation of received datasets/messages coming from the SUT. This allows pairing of stimuli and feedbacks and facilitates statistic evaluation of reaction times.

Alignment of feedback latency
The HILSF shall provide means for alignment of the simulation models' feedback latency by variable artificial delays of transmissions.

Notes
Some periodic task on the side of SUT may generate stimuli and send them to a subsidiary unit controlling some part of vehicle technology or – during testing – to a simulation model implemented by HILSF. The considered task can further read available datasets/messages within a short time interval after sending stimuli and start processing of the obtained data again. In these cases, minimal feedback latency of the simulation model may lead to random skipping between two values of the overall transport delay. Higher computing power of the HILSF in comparison with an original unit may cause greater uncertainty of the behavior in some cases. This uncertainty may lead to worse fidelity of the simulation model and worse conditions for repeatability of tests.

Transport delay modelling
The HILSF shall provide means for modelling the overall transport delay in control loops including time-varying and also pseudo-randomly scattered values of delay.

Notes
The overall transport delay in a control loop includes a part due to physical properties of a plant modelled by HILSF. Another part is caused by the asynchronous execution of all

involved tasks including the transmission of datasets/messages through communication channels of the SUT.

Software portability
Software means of HILSF shall be usable in different environments.

Topology definition
The HILSF shall provide a Graphical User Interface (GUI) to define the topology of the HILSF devices. In addition, it can be used to describe the topology of the given SUT including the descriptions of all communication channels shared by the SUT and the HILSF.

Basic data types support
The HILSF shall support all basic data types defined in IEC 61375-1.

Import of custom data type declarations
The HILSF shall be capable to import text files containing declarations of data types applied on the side of the given SUT. Partial and successive import shall be enabled.

Notes
XML should be the preferred format. In addition, the declarations of the data types applicable in an SUT should be exported partially and successively by one or more IDEs which are exploited for programming several units of the given SUT.

Import of dataset and message descriptions
The HILSF shall be capable to import text files containing descriptions of datasets and messages to be exchanged between the SUT and the HILSF. Partial and successive import shall be enabled.

Notes
XML should be the preferred format. All necessary addressing attributes and structure descriptors of datasets and messages based on already declared data types shall be imported before starting the code generation for the HILSF.

Import of dataset and message descriptions
The HILSF shall be capable to import text files containing descriptions of I/O modules of the SUT, their used channels and all relevant configuration attributes and parameters.

Notes
XML should be the preferred format. A set of the SUT's I/O modules should be delimited according to a given interface between the SUT and the HILSF. Further I/O modules can be applied inside of the SUT. In addition, a set of parameters of I/O modules and channels should include parameters describing their dynamic properties as well as parameters regarding maximum allowed load.

Visualization of configuration data
The HILSF shall provide the possibility to represent all configuration data regarding interfaces to the given SUT or several subsets of them determined by several filtration conditions.

Generation and export of aggregate configuration report
The HILSF shall be capable to generate aggregate configuration report as text file comprising all merged configuration data obtained by successive importing of configuration data.

Endianness conversion
The HILSF shall be capable to convert endianness of incoming and outgoing data according to various formats. Those can be the own endianness and alternative formats standardized by various communication protocols. Another possibility would be native formats for directly or transparently connected processor units of the SUT. All these necessary conversions shall be performed automatically based on the topology description. It involves channel

descriptors, declarations of structured data types and declarations of individual datasets/messages to be exchanged between given SUT and HILSF.
HILSF shall support network byte order (big-endian) for the data exchanged by TCN and other IP network protocols.

2.2.5 Timing design goals

Real-time support for HIL

The simulation and validation framework must support real-time data-transfer to support HIL simulation. Even if the underlying communication network of the framework is non-deterministic such as the Ethernet or the Internet, message delivery and the execution of tasks in time shall be enabled.

2.3 SIL frameworks

Software in the loop (SIL) is a technique where a simulated plant and a simulated controller are interconnected and run in real-time [60]. The technique is widely used and there are different approaches dealing with it. In this section, some examples are presented that are related to the railway (see section 2.2.1) and the automotive (see section 2.2.2) domains as well as distributed SIL environments (see section 2.2.3).

2.3.1 Railway

In the railway industry, algorithm and software development is executed in company-wide standardized tool chains. The major factor influencing the tooling – and thus the simulation capabilities – is the certifiable series code production, not the state-of-the-art simulation.

Thus the tasks of the SIL are part of the software IDE, using the test framework that integrated it. Examples are module and integration tests. This approach is feasible only as the test sequences include the whole plant behavior in an abstract manner. As a drawback, this abstraction prohibits the behavior exploration of behavior across the development focus of one development team.

In the railway industry, the focus lies on automation and reuse of the test cases across software releases and system generations. This is visible in the market by spread use of test tools, e.g. [35] and [46] which have been certified for railway related process standards (e.g., EN50128 [21]).

For functions where the difficulty is related to the physical plant to be controlled, model-in-the-loop setups are used, e.g. [12] and [81]. Such steps will be the first opening the development chains towards model based IDEs. Those are often used in the automotive domain – and therefore allowing the usage of automotive SIL environments.

All in all, the number of SIL frameworks in the railway domain is quite limited and more frameworks considering HIL can be found. This is one research gap this work package focuses on in future work of the distributed co-simulation framework. To reach this, experience from other domains can be exploited. Hence, the focus of the next section lies on the automotive domain where SIL is widely used.

2.3.2 Automotive

In the automotive domain, simulation has a long history and is wide spread in use for the development of software based controllers. Until recently, engineering teams have been free to use the simulation platform of their choice. This situation led to a vast variety of competing general and specialized COTS tools as well as in-house solutions.

Already successful simulation platforms are those for niches driven by outstanding singular requirements. Examples are EB Assist ADTF [2] and TISC by TLK-Thermo GmbH. The former example is for simulation and testing of driver assistance functions. In those use cases, massive data streams from radar and visual sensors are the basis for testing assistance functions which became the core feature for ADTF. However, the latter example simulates thermodynamic systems like automotive cooling.

Since approximately 10 years, general (co-)simulation frameworks are used in automotive engineering, however with limited spread until now. Examples are EXITE ACE by EXTESSY AG (now part of PTC Inc.), Silver by QTronic GmbH, Icos by Virtual Vehicle Forschungsgesellschaft mbH or xMOD by D2T GmbH. Two major drawbacks hindered the wider success. On the one hand, the tools did not follow a standard thus risking the vast investment in models of the tool user. On the other hand, the established processes did not enforce the close cooperation during development across department or company borders.

The situation has changed. First, with FMI [9] a first universal standard for co-simulation and model-integration for signal-based simulation has been introduced in 2010 (see Section 4.3.1). Extensive tool support by more than 90 tools and many industrial projects led to continuous development of the standard.

More importantly the business processes in automotive development are changing gradually. Cost reduction requires more virtualization and the reduction of prototype vehicles. This also applies inside the cars. Functions are decoupled from the hardware architecture as they grow with each car generation and the number of controllers has found a technical maximum already. Thus functions are repartitioned among the controllers requiring very flexible SIL testing without hindrances of company borders.

At present, the success of virtual cooperation by simulation during development is not a technological challenge on the controller side anymore. The following advances are required by the user of the test framework. The first aspect are methods for developing and integrating plant models which perform sufficiently in the context of the different test cases. Methods for managing the vast landscape of models are the second aspects. This includes the versions and the variants of functional requirements and models

2.3.3 Distributed SIL environments

Nowadays, it is common to have distributed control systems with several plants and controllers which are interconnected by real-time communication systems. To simulate those systems, a simulated plant and a simulated controller can be interconnected by a communication network and run in real-time. This technique is called Distributed Software-In-The-Loop simulation (DSIL). In the following, two different environments for DSIL are discussed. While the first approach realizes a distributed real-time simulation environment called CEMTool, the second one is related to discrete event simulations. It is called SPADES (System for Parallel Agent Discrete Event Simulation).

Real-time distributed SIL simulations are not simple to realize. They require a scheduling algorithm to reduce network delays and guarantee real-time behavior. Additionally, a Computer-Aided Control System Design (CACSD) tool is useful for quick controller design and the simple connection of various hosts. It is effective for reducing development costs and the amount of trial and error in the development [60].

The authors of [60] developed such a CACSD tool for real-time DSIL which is called CEMTool. Their requirements are (i) easy network handling, (ii) the tool should design various control algorithms easily, (iii) quick controller design and (iv) a compatible programming grammar for all kinds of network scheduling algorithms. Furthermore, the network must consider some control-oriented requirements. In each sampling interval, data transmission and the computation of the control algorithms must be completed (i). Additionally, a network scheduler must reduce network-induced delays (ii) and the network

must be reliable (iii). Finally, data for a given time must be processed together and should not be missed with data for different times (iv).

In CEMTool, the nodes are connected via an Ethernet network. Since Ethernet uses a random media access protocol without determinism, the environment requires a scheduling algorithm to provide real-time guarantees. This algorithm works according to the master/slave concept. The node with the smallest network load and computation time is chosen as the master. A real-time clock in the master creates interrupts at the beginning of the periods but there is no synchronized global time. Hence, the sampling intervals of all nodes must be the same. As soon as an interrupt occurs in the master, it informs the slaves. Afterwards, the master computes its control algorithm and sends its data to its partners in the next time slot. The partners are waiting for the data and they perform their operation when the data arrives. At fixed time-slots, the slave's output data is sent to their partners.

SPADES [84] is a discrete event simulator for various applications in artificial intelligence research. Its fundamental simulation components are agents whose thinking times are tracked and reflected in the results. For this, the simulator uses software-in-the-loop supporting distributed execution across multiple systems. Furthermore, the results are reproducible with no influence of network or system load. In SPADES, the environmental parameters can be tuned to execute the large number of trials required for machine learning. Moreover, it provides interoperability as long as the agent architecture as well as the language has the capability of writing to and reading from Unix pipes.

A typical execution cycle in SPADES consists of sensing, calculation and actuation which can be executed as a pipeline. The thinking time is assumed to be non-negligible such that it has to be included in the simulation. Moreover, the time varies based on the inputs. However, the thinking time is the same as in the real execution since the deployed software is included in the simulation. Measurements of the Central Processing Unit (CPU) time required to process an action are performed based on a modified kernel counting CPU cycles.

Much work in distributed simulations is related to break down the simulations into executable components so that communication requirements are low. Instead of using a flexible and adaptable organization, the breakdown is fixed in SPADES. It allows as many agents as possible to compute without violating causality. For this, SPADES uses a conservative discrete event simulation algorithm. The algorithm only processes events if the causal order is met. However, events can be processed out of temporal order if they are not causal related. Interactions of agents in the environment are not necessarily synchronized due to the discrete event nature. Any set of agents can perform actions at a given time-step. Hence, smaller time-steps for the simulation do not increase the simulation network's load and all actions are realized at the correct time.

The architecture of the SPADES environments consists of a world model connected to a simulation engine and multiple agents. In this environment, the world model represents the environment to simulate and the simulation engine is implemented in C++. As the center of the discrete event simulator, the simulation engine queues all events. It acts as a master and coordinates the communication. Queuing the events is realized by a centralized list. Each process that needs to schedule an event must notify the master. This notification is implicitly realized by the agent's actions. Hence, the master has complete knowledge about pending events at all times and can determine which events can be processed safely. Furthermore, a communication server is connected to each agent. It communicates with the simulation engine using the TCP/IP protocol stack. However, the agents and communication servers exchange data by Unix pipes. In that way, the agents can be implemented in different languages as long as they support those pipes. Major drawbacks of this implementation are the lacks of efficiency and scalability. Since the master coordinates the communication and manages events, it can become a performance bottleneck slowing down the simulation.

2.4 HIL frameworks

Software-based simulation cannot exactly replicate real operating conditions [68]. One solution to overcome this problem is the replacement of simulated models with real hardware devices. The subsequent sections present different approaches in the railway (see section 2.3.1), the automotive (see section 2.3.2) and other domains (see section 2.3.3). Furthermore, there are available environments for distributed HIL simulation depicted in section 2.3.4.

2.4.1 Railway

A HIL can be realized following two approaches, c.f. the figure below.

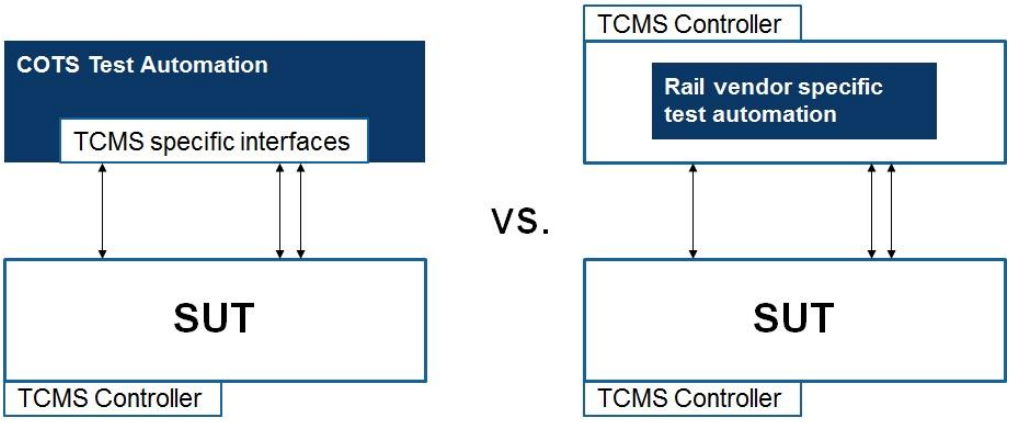

Figure 2.10 Two approaches for realization of the HIL

In the automotive domain, COTS test automation and HIL hardware are typically used. This requires a TCMS specific interface to be developed for the connection to the SUT. As all railway vendors use their individual TCMS controllers without standardized interfaces this approach is most often not followed.

For the sake of an efficient and easily usable HIL simulation, railway vendors tend to use the same TCMS controller as HIL environment as the SUT. All interfaces are inherently available. This leaves the task of implementing rail vendor specific test automation in the controller service as HIL.

This implementation can be done using the same tool chain as the SUT. Thus tool chain certifications are inherited and the engineering know-how is already available, the effort for establishing HIL testing is reduced.

Besides this, there are various HIL frameworks available for the railway domain. Facchinetti and Bruni study the interaction between a physical pantograph and a numerical model of a centenary. Their test bench reproduces the interaction in a 0-20 Hz frequency range including the effect of stagger in the contact wire [105]. A HIL simulator which connects a vehicle control system to a real-time dynamic vehicle simulator can be found in [106]. The authors focus on the simulation problem's hybrid nature and causality variations between discrete and continuous parts. The authors of [107] implemented a system to simulate the traction system of an automatic subway. It is based on two induction machines representing the influence between a mechanical power train and traction drives. In another work, they exploit HIL testing to validate an anti-slip control mechanism for traction systems. Finally, Baccari et al. use HIL to test the control software of electromechanical train components [109]. They use mathematical models to represent relevant electromechanical components of a power train and test them against the control software of related ECUs.

2.4.2 Automotive

The automotive industry is dominated by a number of HIL vendors: dSPACE, ETAS, National Instruments, OPAL-RT or Vector Informatik. All of them have their specific strength but are characterized by being largely not interoperable with each other.

One example for such a tool is presented by Kulkarni et al. [58]. The after-treatment system in engines controls the environmental pollution due to NO_x particles in exhaust gases. To improve testing time and to improve the results, Kulkarni et al. introduce a HIL system. This system is built on an engine emulator called Load-box User Interface System (LUIS Bench) and FMET box. It uses NI TESTSTAND and NI LabVIEW to automate the Failure Modes Effects Tests (FMET). The developed system simulates the required fault conditions. According to the testing requirements of FMET, it removes the fault conditions and reaches the same results as the conventional bench FMET. However, the simulation requires significantly less time.

Under the guidance of the ASAM association, several standards have been developed bridging these gaps of interoperability. The ASAM Standards cover all interfaces necessary in an extensive HIL setup – with one exception.

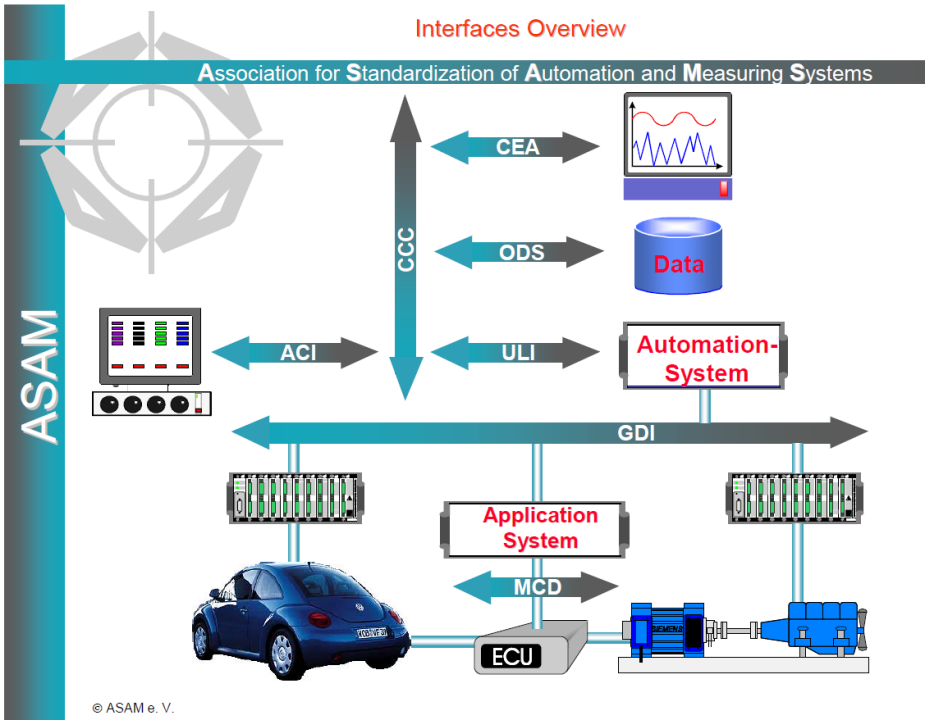

Figure 2.11 Overview of ASAM Interface Standards [50]

In 2013 the ASAM XIL API was published for the connection of different components of a distributed co-simulation (ASAM - Association for Standardisation of Automation and Measuring Systems [33]). It became apparent that the ASAM XIL API is closely related to the FMI Standard which is explained in detail in Section 4.3.1.

The similarity of test execution on HIL using models and co-simulation of SIL has led to an effort joining the two streams into one. This lead to the ITEA3-funded consortium ACOSAR (ACOSAR Consortium [24]). Their work shows an extensive overview of material and requirements of the whole domain which shall not be repeated here.

2.4.3 Other domains

HIL frameworks are not only used in railway and automotive, but also used in other domains. Examples are power quality studies in NAVY all-electric ships (see [66], e.g.), electrical grids

(see [79], e.g.) and space research (see [97], e.g.). The referenced approaches are described in the following.

The platform provided by Liu et al. [66] is utilized to test the sensitivity for power quality deviations of a variable speed drive controller card. To achieve better accuracy of the testing results, the authors implemented a method based on a real-time HIL simulator. It is composed of a digital simulator (RTDS), the tested hardware as well as their interfaces. Successful experiments of a U.S. Coast Guard icebreaker have contributed to the conceptual design of a universal power quality test bed.

Experimentally evaluating control strategies for power interfaces require a significant allocation of time and other resources [45]. Power HIL (PHIL) simulation can support the process, hence the authors of [45] demonstrate a micro-grid test bed for Distributed Energy Resources (DER). They combine PHIL techniques with real-time systems by providing a hierarchical, model-based approach. This approach significantly improves flexibility by configuration. Due to its hierarchical character with RT targets and RT-embedded industrial controllers, it applies real-time control. Hence, it allows to analyze the performance of grids, micro-grids and DERs. Experiments with two use-cases show the approach's versatility and usefulness for reproducing realistic tests.

Paolone et al. [79] present a HIL proof-of-concept of a real-time state estimator (SE) for Active Distribution Networks (ADN). Their goal is to reach high-rates of power network state assessment. Hence, the time-scales for the execution of the estimation should be in the order of microseconds. To reach this, data is acquired from Phase Measurement Units (PMUs), aggregated in Phasor Data Concentrators (PDCs) and stored in a real-time database which is coupled with the estimation. Since the real state is not known exactly, Real-Time Simulators (RTSs) are used to calculate an approximated state. This approximation is compared to the estimated state to determine the accuracy of the state estimation process. Experiments in the ADNs have shown acceptable accuracy levels according to the operation conditions.

The Alpha Magnetic Spectrometer (AMS) was mounted on the International Space Station (ISS) to search for dark matter. Since there are difficulties in ground testing due to hardware and environmental limitations, Sun et al. introduce a time-saving approach [97] in HIL simulation that connects real controllers with virtual devices and reuses flight data instead of mathematical models. The AMS consists of various detectors whose data is processed by controllers in the system. From the ground, manual control is possible if there is a good communication connection. However, communication outages pose potential risks. In the HIL simulation, a master node controls its slaves by sending requests to them. As there are many nodes, they all cannot be simulated exactly using mathematical models. Hence, the slaves send real data from the AMS in space. Real components are connected to the simulated ones using an interface device. This interface realizes the communication using appropriate protocols for the use-case. Experimental results show the effectiveness and the efficiency pf the approach.

2.4.4 Distributed HIL frameworks

HIL simulation combines the accuracy of physical prototyping with the cost effectiveness of model-based simulation. Furthermore, integrating multiple HIL setups allows the full exploitation of the HIL benefits. Those are reasons why recent research has focused on reaching this integration over the Internet as Distributed HIL Simulation (DHILS). [28]

One desired and important concept in DHILS is transparency. It measures the accordance of the distributed system's dynamics to that of directly coupled tools. If the value is high, the dynamics do not change significantly. [28] presents different metrics which measure transparency and a statistical approach based on time-domain analysis. This approach even handles nonlinear stochastic systems as present in Internet-distributed HILS. Two sources of transparency degradation in the Internet are Quality of Service (QoS) and distributed real-

time simulation. Aspects of QoS in the Internet are delay, jitter and packet loss. While delays are related to routing and processing in the network, jitter is the variability of the delay. Packet loss occurs due to the Internet's best effort character. However, those sources are well known and still subjects to research. Distributed real-time simulation mainly requires co-simulation in real-time with data exchange at defined coupling points (see Chapter 4). The communication over the Internet introduces sampling effects which will degrade transparency. Without having access to the state information of all nodes, the theoretically established accuracy is not achievable and hence transparency will be reduced.

The second goal in [28] is to identify the major cause of transparency degradation through research on accuracy reduction. Using this knowledge, mechanisms to increase transparency can be developed and fundamental limitations can be indicated. As a third goal, the authors examine if transparency depends on output signals in the system or if it is an independent property. Experimental results show that distribution has an adverse and significant effect to transparency which can even dominate QoS aspects. Thereby, there is no difference between Local Area Networks (LANs) or the Internet as long as the latter incurs small delays. However, there are different use-cases where the effect of distribution is insignificant. This shows that transparency is not an independent property but depends on the signal of interest.

Lu et al. [68] present a high performance real-time simulation environment to obtain high-fidelity results in HILS. They extended the Virtual Test Bed environment (VTB, see Chapter 4) by a real-time component (VTB-RT) to couple a simulation environment with the hardware under test. VTB-RT is a low-cost hard real-time simulation environment built on open-source software and off-the-shelf hardware. It maintains acceptable results and even supports distributed simulations. Since VTB-RT shares major parts with the underlying VTB environment, both tools are compatible to each other. The environment utilizes the open-source real-time application interface which is a Linux kernel modification. This interface modifies the handling of hardware interrupts and enables the Linux hard real-time capabilities. As a HIL simulation requires I/O interfaces to hardware, the Comedi freeware is used. It provides device drivers for many data acquisition tools and works with the Linux kernel as well as the Real-Time Application Interface (RTAI). To solve the models, the SRC solver is utilized which manipulates the real hardware interfaces. As a result, it enables the coupling of the hardware plant with the simulation environment.

CORESIM [62] is an approach for realizing interoperability between non real-time and real-time simulations in a plug-and-play approach. Such systems require a synchronization mechanism of the logical time and the real time. There are two different mechanisms, a lock-step and a time-slicing approach. The first one is easy to implement but slows down the simulation because every simulation tool processes one simulation step and waits for the other tools. Since the real subsystems are frozen for short times in this approach, it is not useful in use-cases like velocity control. Opportunistic strategies exploiting time-slicing use longer intervals in which the real subsystems are frozen. In those intervals, the simulation tools catch up at their own speed having a copy of the logical time. If the tools require new data which is not available yet, proxies are used to approximate the data. As soon as the correct data is available, it is compared to the approximated data and the simulation is rolled back if necessary. This approach has some disadvantages. For example, it is possible that the logical time continually rewinds to the beginning. Hence, the authors of [62] use fixed intervals for the synchronization and fast proxies with rollback. Using this, the real hardware can run for a reasonable duration of time. Furthermore, there should be only one rollback per period. However, interactions between the subsystems are not properly modelled and the ideal period length is a balance of simulation accuracy and the real subsystem's run-length.

2.5 Research gap for state-of-the-art

In the context of HIL and SIL frameworks, a research gap can be identified in the following challenges. Those are related to the handling of disturbances, setting up the HIL and the temporal behavior of HIL and SIL simulations.

The first challenges cover the handling of distributed simulation in case of disturbances and efficient simulation set up. Disturbances can delay the communication in the framework. Hence, it shall provide automatic functionality for assessing the simulation for delayed communication and for a structured failure reaction while it still ensures a valid simulation behavior. This task is similar to the development of a failure reaction and central reaction handler of a car. Moreover, not every disturbance is of equal severity for the consistent behavior of the simulation. The simulation framework shall provide a sensitivity analysis for disturbances occurred during the simulation for assessing the simulation validity online or in post-processing. Due to timing constants inside the simulation components, the timing architecture of a composed simulation differs from the timing of the communication links between the components. To realize the efficiency of the simulation setup, an automatic system analysis is required. It calculates a simulation setup from the composed system description and the simulation models properties.

Setting up HILSF frameworks should be realized in a hybrid way. To realize HIL testing, two competitive approaches were presented in Figure 2.9, while the second approach is usually preferred by a TCMS vendors/integrators. There are generic modular system which are designed for the composition of TCMS systems. In some cases, those generic systems are not equipped by instruments adequate for the implementation of some partial sub-models of plants. Then, the exploitation of some COTS simulation tools can be helpful. Cooperation of disparate simulation instruments must be solved in such cases. The application of a standardized interface is desirable in this context and the advantage option could be the FMI interface. In this context, adding HIL specific attributes into tool specific parts of the common FMI modelDescription.xml is useful. Those attributes include datasets/messages and variables.

Regarding the timing behavior, the presented HIL frameworks are all capable of providing real-time simulations. Hence, the research gap in this context will be the inclusion of the available real-time concepts for HIL over the Internet.

Chapter 3 Simulation of wireless railway networks

3.1 Design goals for T2G test environment

The simulation framework will be able to perform an early validation of TCMS including its interface to the ground systems.

Nowadays, T2G communications provide information about position and speed of the train in order to ensure the passengers' safety. However, as the popularity of this public transport increases, more efficient and reliable railway systems are demanded. Moreover, the recent progress in mobile telecommunications technology has allowed railway control systems to go a step forward.

3.1.1 Quality of Service (QoS) testing of TCMS T2G interface

This chapter details the QoS parameters for the Train Control and Monitoring System (TCMS) interface to T2G system.

The QoS parameters for a T2G interface depend on the operational applications which rely on this interface. These applications may be classified in [70]:

1. Control applications.
2. Monitoring applications.
3. Video applications.

To specify the parameters, the work from Roll2Rail has been considered [1]. Based on CENELEC EN 15380-4 standard that determines the function groups of a railway vehicle, the functions with the needs of the data exchange over Train Communication Network (TCN) have been identified. In addition, their required communication characteristics have been assessed. These agreed characteristics include among others the data classes, safety relevancy and QoS parameters (data rate, latency and jitter). They have been further mapped to train applications (Functions in the terminology of TCN).

It is to be noted that the TCN serves for the communication of different kinds of applications found on board of a train. It turned out to be beneficial, not only for the requirements elicitation process, but also in general, to group the applications into the following three function domains:

- TCMS safety and non-safety function/functionalities - its functions are mandatory to ensure safe train movement and to ensure carrying the payload (freight or passengers),
- Operator Oriented Services (OOS), e.g. Closed Circuit Television (CCTV) – the functions of the domain are aimed at improving the operational parameters of the train (e.g. maintenance costs, general vehicle availability).
- Customer Oriented Services (COS), e.g. infotainment – the functions of the domain are related to passenger comfort. Here, customers' own devices may be interfaced to the COS functions.

The functions which a T2G communication system handles are shown in Table 3-1, as well as their required communication characteristics (packet size, data rate, cycle time, latency and jitter). Signaling system, voice and video transmissions as well as Passenger Information Services (PIS) data is transmitted. Also, in the context of the new Board to Ground communication system which is defined in Roll2Rail, the T2G link is proposed to send monitoring data about the TCMS to ground to develop new monitoring applications. Although the TCMS functions to monitor have not been defined yet, the QoS parameters have been already fixed**.** In Table 3-1, they are shown for every function. These parameters (data rate, latency and jitter) must be guaranteed by the T2G communication system for a proper working of every function.

Table 3-1 QoS parameters of the T2G functions.

CENELEC Function	**Function domain**	**Packet size**	**Data rate [bit/s]**	**Cycle time [s]**	**Latency [s]**	**Jitter [s]**
Provide train to ground communication	TCMS	128		0.2	1	0.1
Alarming mechanism to the ground	TCMS	512		10	1	0.5
Provide administration service for communication to the ground	OOS		97.65kbps		10	
Send diagnostic data to the ground	OOS		9.53Mbps		60	
Send condition data to the ground	OOS		97.65kbps		10	
Send train position to the ground	TCMS	256		0.5	1	
Send train status to the ground	TCMS	128		0.5	1	
Send voice data to the ground	OOS		62.5kbps		0.2	0.03
Send video data to the ground	OOS		95Mbps		1	0.1
Provide alarming service to the train	TCMS	512	9.76kbps		1	
Provide administration service for communication to the	OOS		976.56kbps		10	

train						
Download software to the train	OOS/COS		95.36Mbps		10	
Send train configuration data to the train	OOS		95.36Mbps		10	
Send diagnostic data to the train	OOS		95.36Mbps		60	
Send PIS data to the train	OOS		95.36Mbps		60	
Send voice data to the train	OOS		62.5kbps		0.2	0.03
Send video data to the train	OOS		95Mbps		1	0.1
Antitheft alarm	OOS	512		5		

3.1.2 Test environment for the testing of the TCMS T2G interface focusing on the ground peer and functionality testing

The T2G Test Environment (T2GTE) will enable testing and validating the T2G system which is specified in the standard IEC 61375-2-6. It observes the complete chain which also includes on-board systems and ground (wayside) systems.

The basic context diagram of the T2G system is shown in Figure 3.1 .

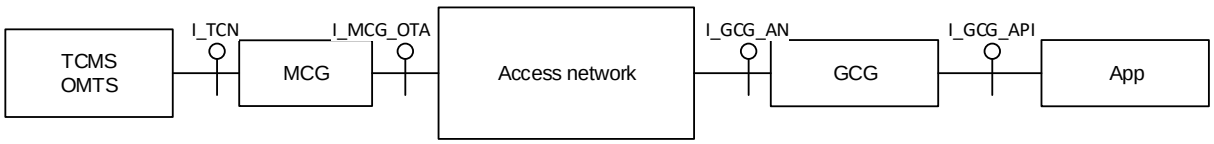

Figure 3.1 - T2G system components and interfaces

Depending on the kind of tests performed, each of the components shown in Figure 3.1 can be a component of the T2GTE, a SUT (System under Test) or a part of it.

TCMS, OMTS represent a whole on-board train system with all its functionalities. The TCN communication network is included in this representation.

MCG is a Mobile Communication Gateway. There can be more MCGs in one train connected to the TCN and up to hundreds of MCGs connected to one GGC.

Access network represents an Over The Air (OTA) wireless communication channel as well as the ground communication infrastructure. It is a general IP-based network and realized as a dedicated railway communication network or as an overlay private network (e.g., Virtual Private Network (VPN)) over the public GSM/3G/4G/5G or WiFi (Wireless Fidelity) network.

The MCG can communicate with the GCG over multiple access networks and select the network offering the best-fit QoS. Furthermore, it is able to switch the networks dynamically as the train moves. It is also possible (but not mandatory) to communicate over several networks simultaneously.

The Access network will always be simulated in the T2GTE, since real wireless networks cannot guarantee the reproducibility of tests.

GCG is a Ground Communication Gateway which provides standardized application services and manages the security layer of the T2G communication.

App is any application which needs to exchange data with on-board systems. One GCG can provide access to multiple Apps.

I_TCN is the standard communication interface between an MGC and other train systems (TCMS, OMTS) provided by means of the TCN. It is specified from the physical layer (Ethernet) up to the application layer (communication services).

I_MCG_OTA are one or multiple physical wireless interfaces. As only the upper communication layers starting from the layer 3 are standardized in the IEC 61375-2-6 this interface can be implemented as any IP based network. In the T2GTE, it shall be implemented as a selected subset of physical networks (e.g. GSM 2G, 3G, LTE), because real HIL devices will be connected.

I_GCG_AN is the communication interface of the GCG for communication with MCGs through access network(s). Since it is specified only on OSI layer 3 and higher, it can be implemented as a generic Ethernet interface.

I_GCG_API is the GCG interface to the external application. It is not standardized and will not be the subject to be tested.

The T2G test environment shall enable the execution of the following groups of test scenarios. The communication between the components will be based on the state-of-the-art of wireless communication networks (see Chapter 3.2).

Tests of the MCG device

MCG test scenarios will cover tests of the MCG device itself. MCGs will be SUTs and will be present as HIL in the T2GTE. Traffic generated by on-board TCMS devices as well as TCN network functions will be simulated on the I_TCN interface. By means of the physical I_MCG_OTA interface, MCGs will be connected through Access network(s) simulator to the simulated GCG which enables testing of the standard OTA services. The T2GTE will also generate network traffic from simulated ground applications.

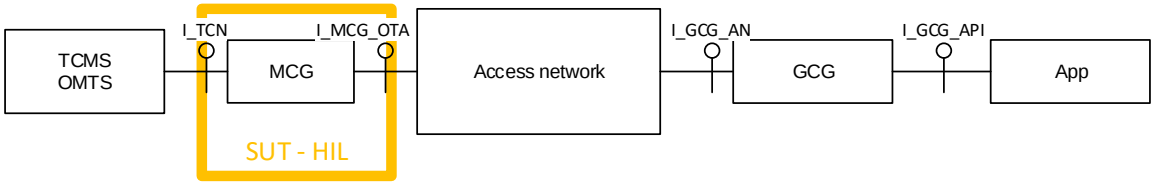

Figure 3.2 - Test setup for MCG tests

Tests of the GCG

GCG tests scenarios will cover software integration tests of the GCG. Since the GCG will be an enterprise IT application based on standard server hardware or even provided in cloud IaaS or PaaS services, we do not anticipate HIL testing here. The GCG will be tested as SIL or as an external service connected by the I_GCG_AN interface to the simulated access network. Traffic generated by fleet of vehicles equipped with MCGs will be simulated by the T2GTE.

The interface I_GCG_API is not standardized and will probably be vendor-specific. The T2GE shall at least provide a framework for realizing mock-up traffic on this interface.

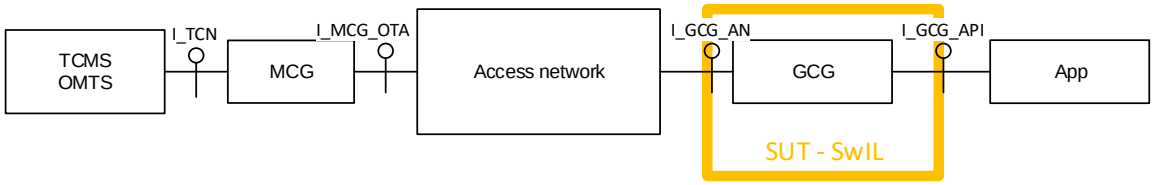

Figure 3.3 - Test setup for GCG tests

Tests of the complete T2G communication system

In this group of test scenarios, integration tests of the whole T2G communication system will be performed. The test setup will combine the setups of the first two groups of scenarios – i.e. both real MGCs and GCGs will be present as HIL or SIL. Both, MCGs and GCGs will be connected in the T2GTE by simulated Access network(s).

Traffic generated by on-board TCMS devices as well as TCN network functions will be simulated on the I_TCN interface. As mentioned above, the I_GCG_API is not standardized and similar to the previous case, a framework for mock-up traffic shall be provided.

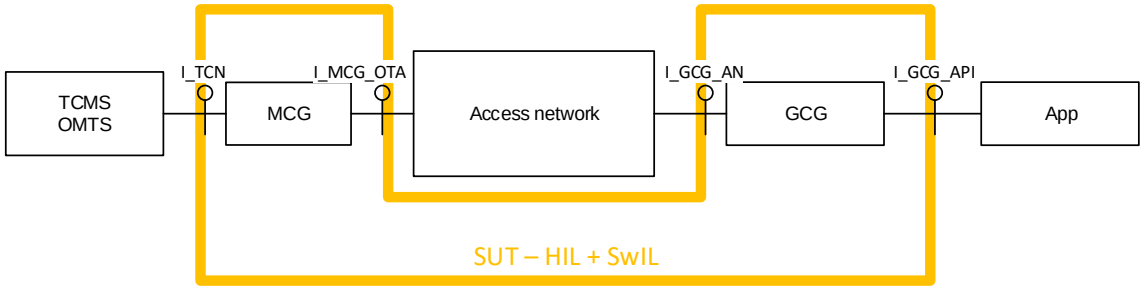

Figure 3.4 - Test setup for T2G system integration tests

Integration tests of the on-board – ground distributed applications

This group of test scenarios will cover an integration test for distributed functions or software applications. This test includes vehicle subsystems and ground subsystems. One Example of such an application is the distribution of route maps for Automatic Train Operation (ATO) system or remote camera surveillance system.

On-board components of the application can be connected to the T2GE directly through the I_TCN interface or it can be part (HIL or SIL) of a simulation of a complex TCMS system. The vehicle subsystem can be present in multiple instances (up to several hundred).

The T2GTE will simulate a complete T2G communication system. MCGs and GCGs can be HIL/SIL or can be replaced by simulation models in this setup. However, access network will always be simulated.

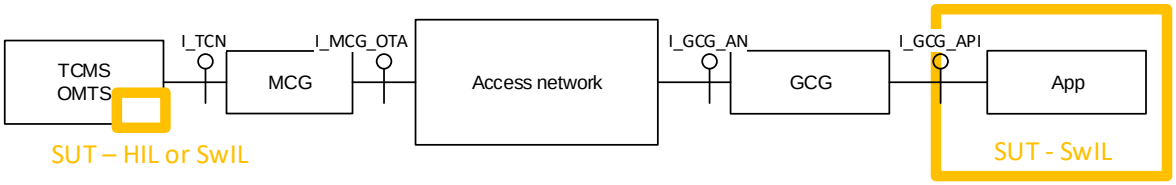

Figure 3.5 - Test setup for integration tests of distributed application

3.2 Simulation of T2G wireless communication networks

In the literature, there are a lot of research works in wireless communications dealing with the simulation of wireless communication networks particularly in the context of mobile communication networks. These simulation approaches can be utilized in a test environment to evaluate the communication between the MCG and the GCG as described in the previous section. We can mainly find four categories of works to simulate the networks:

- simulations devoted to radio channels modelling;
- simulations at physical layer of the communication system dealing mainly with the evaluations of signal processing algorithms;
- simulations on upper layers of the Open System Interconnection (OSI) model (MAC (Media Access Control) or LLC (Logical Link Control) layers for example) to evaluate some specific process such as handover for example;
- simulations of the whole telecommunication network behavior with several base stations or access points and several mobile terminals to evaluate the global behavior of the system versus traffic load for example.

The topic of train to ground simulations (T2G) is more restricted in the literature but one can find a lot of available references. With the same classification we can mention works related to radio channels modelling in the context of urban rail [36] or in the context of a high speed line [102][102]. The aim of these works is to provide radio channel models (path loss, delays, shadowing, *etc*.). These models can be used to predict radio propagation in railways environments or to evaluate the behavior of the physical layer of the system in various environmental conditions.

On the second category, we can mention several papers related to the evaluation of enhanced MIMO (Multiple Input Multiple Output) algorithms at the physical layer. The application is focused on existing CBTC (Communication Based Train Control) system based on WIFI liked modules in tunnels [59, 31]. [51] presents more specific simulations in the context of high speed trains concerning the estimation phase of the signals. The aims of these works were to propose new signal processing algorithms (modulation, coding, MIMO precoding techniques, video coding, channel estimation, *etc*.) at the physical layer to evaluate the performance of these techniques in various channel models representative of the railway environments (tunnels or high speed). The evaluation will focus on the evaluation of BER (Bit Error Rate) or FER (Frame Error Rate) in general for a given signal to noise ratio and QoS when it concerns video. The performances are generally compared to KPI expressed by industry in the case of CBTC or ETCS (European Train Control Systems) systems. Another paper presents a testbed for evaluating LTE (Long Term Evolution) in High-Speed Trains context [86].

We can also mention works on handoff performances in the context of CBTC [49], or the QoS analysis of train-2-ground communication system based on TD-LTE [103].

Other works deal with the behavior of the whole wireless network using OPNET modeler in the railway domain. OPNET is a communication network simulator based on the principle of discrete event simulation. It enables global modelling of the communication systems through a large library of network components, communication protocols, application models and data streams. OPNET modeler integrates various existing telecommunications standards as well as fairly accurate statistical analysis. For example, it allows a complete modelling of a LTE communications system. It models the behavior of the access network (radio) and the core network Evolved Packet Core (EPC). It is also possible to test the mechanisms of QoS, management of the mobility, the adaptability of the radio profiles, *etc*. It allows taking into account radio channel models.

Sniady & Al present some evaluation of LTE performance on the context of high speed line for different scenarios [93, 94], while Aguado & Al. have considered WIMAX and LTE [3, 67]. Kassab & Al. have implemented a LTE Based Communication System for Urban Guided-Transport and they present a QoS Performance Study [55] in different architecture scenarios versus the number of trains. Furthermore, Sondi & Al. present how to consider test scenarios based on real-world traces for the ERTMS Telecommunication Subsystem Evaluation [95].

In addition to OPNET, other discrete event simulators have been used to study the behavior of a whole wireless network for T2G. OMNeT++ has been used to analyze a WiFi network for CBTC in urban trains in [104]. Jointly, a microscopic railway simulator (BRaVE) is considered

to model the basic functionality of railway control and management. A hybrid WiMAX-WiFi solution is simulated in [22]. The proposed communication system uses a WiMAX interface for the T2G and a WiFi interface for the intra communications. The application services provided is VGA video and throughput, packet losses and latency are evaluated in a handover scenario inside a tunnel for two different velocities. Maureira & Al. have developed a simulator for GSM and WiFi networks in OMNeT++ [71]. The simulator allows testing several algorithms in different environments since several physical parameters such as Doppler Effect or fading channels can be configured.

Furthermore, NS-2 is used by Casasempere & Al. to evaluate video transmission over WiMAX [19]. Several mobile users generate the H.264 video traffic flow and send it to a base station. The delay is evaluated in function of the number of nodes and the speed of them. [77] uses NS-2 to simulate WiMAX and WiFi networks for public transport. The Internet connection is simulated in two cases, off peak hours and peak hours, where the number of passengers trying to get Internet access differs. Also, satellite systems have been simulated using NS-2 [69], and the electrical trellises effects are evaluated for TCP applications.

Recently, Nguyen & Al. have presented a new analytical approach to evaluate the critical-event probability due to wireless communication errors in the T2G link for Train Control Systems [76]. They have considered Petri Nets model.

All these research works are summarized in Table 3-2.

Table 3-2 Research works of Train to Ground (T2G) simulations

Ref.	**Technology**	**Scenario**	**Simulator**
[102]	Radio channel models	Urban train	Measurements campaigns and simulations based on ray tracing techniques
[36]	Radio channel models	High speed train	Measurements campaigns and simulations based on ray tracing techniques
[59]	Physical layer	Urban train	Measurements campaigns and simulations on signal processing
[31]	Physical layer	Urban trains	Simulations on signal processing
[51]	Physical layer	High speed train	Simulations on signal processing
[49]	Handoff	Urban train	Simulations
[103]	TD-LTE	High speed train	QoS analysis
[86]	Testbed LTE	High speed train	-

[93]	LTE	High speed train	OPNET
[94]	LTE	High speed train	OPNET
[3]	WiMAX	High speed train	OPNET
[67]	LTE	High speed train	OPNET
[55]	LTE	Urban train	OPNET
[95]	ERTMS test scenarios	High speed train	OPNET
[104]	WiFi	Urban train	OMNeT++
[22]	WiMAX-WiFi	-	OMNeT++
[71]	GSM and WiFi	Urban train and high speed train	OMNeT++
[19]	WiMAX	-	NS-2
[77]	WiMAX and WiFi	Public train	NS-2
[69]	Satellite system	-	NS-2
[76]	Train Control Systems	Urban train	Analytical (petri nets model)

3.3 Research gap for state-of-the-art

The research gap for the simulation of wireless railway networks lies in the development of a test environment for testing the Mobile Communication Gateway (MCG) and the Ground Communication Gateway (GCG) in T2G communications. For this, the state of the art of Simulation of T2G wireless communication networks and the description of the test environment for the testing of the TCMS T2G interface described in chapter 3.1.2 must be taken into account. The MCG and the GCG, which are the Devices Under Test (DUTs), provide the interface between TCMS and ground systems through a wireless network, as shown in Figure 3.6.

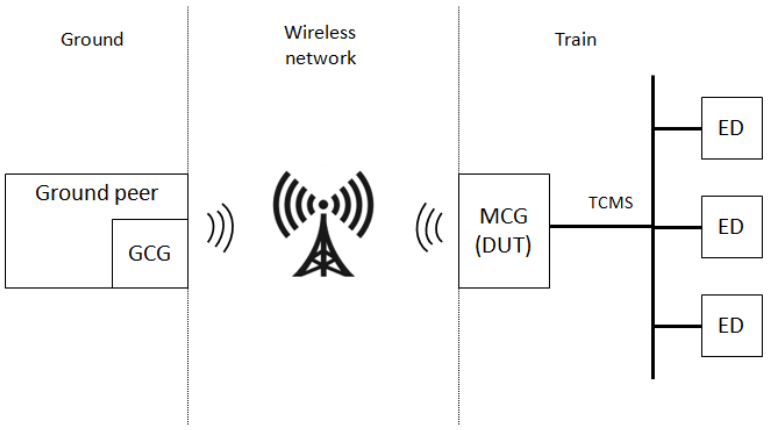

Figure 3.6 T2G and TCMS networks in a train.

The test environment will be based on the use of the OPNET discrete event simulator from Riverbed. It simulates the TCMS and the wireless network.

Depending on the test scenario (see chapter 3.1.2), the MCG may be a real product connected to the simulator using HIL or SIL. The MCG will be a HIL device in those scenarios in which it is considered as DUT and a SIL device in those scenarios where the DUT is the GCG. The GCG will always be tested as SIL. The possible simulation environment is shown in Figure 3.7. The MCG is developed in the Roll2Rail context, while the GCG as well as the test ground application will be provided by the CONNECTA project.

Some End Devices (ED) and their application, which sends and receives data from the ground peer through the MCG, will be simulated. In the wireless network simulation, a channel emulator will model the physical layer for one link on different technologies (LTE, GSM, UMTS, WiFi) and the real wireless channel. A channel emulator can be used to emulate physical channel characteristics. If we will not use the emulator, different propagation features could be configured (delay spread, channel attenuation, etc.) with OPNET on a very simple basis. Regarding the different network characteristics and the possible scenarios, they will be simulated by the discrete event simulator, whenever this is possible. A thorough analysis of each scenario, matching the simulator capabilities, needs to be previously completed.

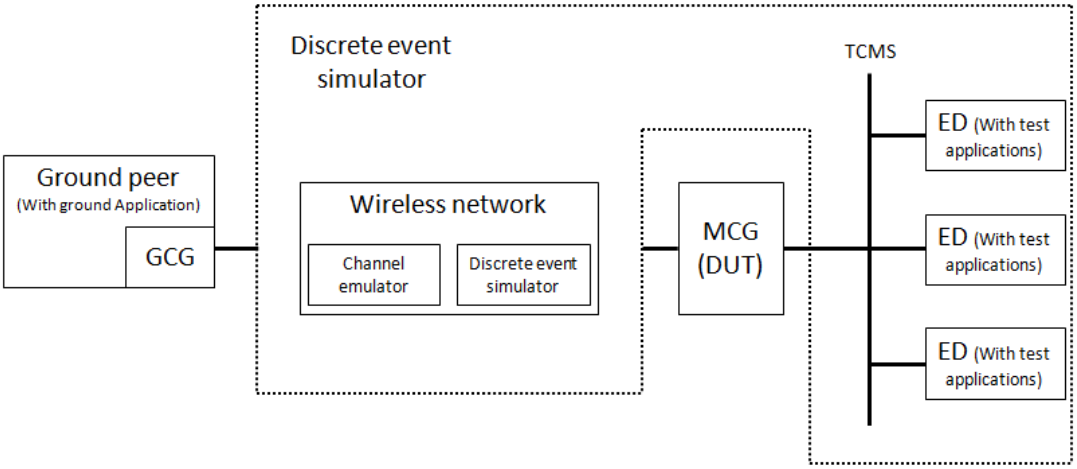

Figure 3.7. Simulation environment for MCG tests.

In OPNET we can consider a very simple GSM model, some Wifi models and a quite accurate LTE model. The LTE model is following the standardization and is considered as the best one for discrete event simulators. Each fixed network is associated with appropriate mobiles that are able to operate only with a single network. In the project duration, it will be difficult to implement several standards; thus, a choice should be done to keep realistic, attainable targets.

Only in simulations, we can model a MCG as a mobile equipped with several physical interfaces which is desired for the real system. A specific applications layer (middleware) could be developed to manage the heterogeneous handovers between the different networks. However, it will not be possible to simulate this behavior without the knowledge of how the MGC manages the handovers.

Chapter 4 Distributed simulation frameworks and co-simulation

4.1 Design goals for the distributed co-simulation framework

In this section, design goals for the distributed simulation framework are defined. Those are separated into design goals for co-simulation (see Section 4.1.1) which are further extended by design goals for distributed co-simulation (see Section 4.1.2). Additionally, there are design goals for applicability (see Section 4.1.3), timing (see Section 4.1.4), configuration and reconfiguration (see Section 4.1.5) and security (see Section 4.1.6).

4.1.1 Design goals for co-simulation

Network-centric simulation at system level

The simulation tool focuses on the communication in the train-network. Furthermore, the degree of abstraction lies on the system level using models that are implemented in a high-level language.

Discrete event simulation

Since the focus of the simulation framework lies on the communication network, the progress of the simulation will be performed in a discrete manner.

Determinism and reproducibility

Determinism and reproducibility are desired since a repetition of the simulation with the same inputs and the same configuration will produce the same results. In this way, simulation executions with different parameters are comparable and testing is simplified.

Interoperability between heterogeneous (communication) systems and protocols

By providing interoperability, the co-simulation framework is unrestricted to the simulation of defined communication systems and protocols. Those become exchangeable and the framework can be used for all components in the railway domain.

Coupling of different domains

End-devices such as plants usually perform their execution in the continuous-time domain. However, the communication between components in the system as well as the execution of the control algorithms have discrete dynamics. Hence, the framework must provide interfaces and mechanisms to couple both domains.

Portability between different host platforms

The co-simulation may be executed on heterogeneous systems with, e.g., different operating systems, network connections, etc. Hence, the framework should be portable between different host platforms.

HIL and SIL co-simulation for virtual placement in the market

One goal of the Safe4RAIL-project is virtual placement in the market of complex railway systems. To reach this goal, SIL and HIL are required and provide an early validation of the components.

Detection and handling of events by simulators

The simulation tools shall detect events occurring in the simulation to inform the coordination algorithm. Furthermore, an appropriate event handling mechanism has to be implemented.

Synchronization and data exchange between simulation tools

In the framework, various simulation tools with potentially heterogeneous behavior are coupled. All tools must be synchronized in the progress of time to accurately exchange data.

Interfaces and protocols for coordination

The synchronization algorithm requires specified interfaces to the simulation tools to coordinate them and a protocol is required to define coordination messages sent to the different tools.

Generic interfaces to local simulators

Generic interfaces to the local simulators enable the co-simulation of various simulation tools with the same framework. The simulation tools must only implement the interface.

Adaptable synchronization and communication step-size

The co-simulation is performed step-wise. Thereby, the accuracy and the performance of the simulation depend on the step-size of the synchronization algorithm. An adaptable step-size depending on the requirements of the use-case can improve both, the accuracy as well as the performance.

4.1.2 Additional design goals for distributed co-simulation

Support for co-simulation requirements in distributed systems

The distributed co-simulation framework must support all the requirements defined for co-simulation.

Distributed event calendar

The distributed event calendar is used to synchronize the simulation tools. Global knowledge about the occurrence of events simplifies the procedure.

Support of remote procedure calls

The support of remote procedure calls enables the remote execution of functions in other simulation tools.

Scalability

By providing scalability, multiple simulation tools can be easily integrated into the framework. There is a reasonable overhead in integration effort, communication load, etc.

4.1.3 Design goals for applicability

Modular components

Modular components with defined interfaces can be easily integrated and exchanged. Furthermore, the components can be considered as black box and the implementation must not be known.

Re-usage of existing models with minimal changes

Supporting reusability of existing models with only minimal changes facilitates the integration since manufacturers and suppliers can use models defined for other co-simulation frameworks.

Compatibility with existing standards and interfaces

There are already different standards for co-simulation available such as FMI. Compatibility with the standards enables the usage of already existing tools compliant with the standards.

Hierarchical modelling

In the train network there might be components which are composed by networked subcomponents. Using hierarchical modelling, the subcomponents can be co-simulated with the rest of the train.

Expandability and adaptability of the framework

The architecture of the framework shall be expandable by further capabilities and possible future simulation and validation techniques. The integration shall be easily assimilable.

4.1.4 Timing design goals

Synchronization of simulations using a global (simulation) time

Distributed control systems are usually based on a synchronized global time. If the simulation time is also globally synchronized, the real-time behavior of the co-simulation is closer to that of the real system.

Different time resolutions of simulations

There might be simulation tools with different time granularities. Those tools must be connected and synchronized with each other as well.

Event management with correct temporal order

If events are not processed in the correct temporal order, the behavior of a distributed system might not be consistent. Hence, the framework must ensure the correct timely execution.

4.1.5 Configuration design goals

(Re-) configuration at run-time

To provide different use-cases for testing, the framework must be configurable at run-time to prevent recompilation. Since the composition of trains is not fixed at run-time, reconfiguration mechanisms useful in addition.

Configuration by files

The configuration of the subsystems in the framework is realized using files which save the configuration data. Those files are all read by a central instance and the subsystems are configured afterwards according to the data. This procedure automates the configuration process since the user only has to trigger the (re-) configuration.

Interface for (re-) configuration

The components and the co-simulation framework must provide interfaces for (re-) configuration.

Protocol for (re-) configuration over the network

A protocol for (re-) configuration over the network enables the (re-) configuration by a central instance which sends the configuration data to each component. The protocol defines the data-types.

4.1.6 Security design goals

Confidentiality

Assures the non-disclosure of information (e.g. simulation data) towards entities, such as users, processes or devices, unless they have been authorized to access the information. This implies that no one is permitted to access or read the information/data except the dedicated and authenticated receiver entity.

Authenticity

Represents the entities' property of being able to be verified and trusted. The authentication, regarded as a process of verifying the entities' identity, is the assurance that an entity is indeed whom it claims to be. The authentication process includes not only the verification of an entity, but also the verification of a source and the related integrity of data.

(Data) Integrity

Assures that information/data has not been modified, whether intentionally or unintentionally. The assurance of non-alternation implies that the information/data since creation (either in transit or in storage) has not been undetectably modified.

4.2 Common aspects related to co-simulation

In this section, an overview about co-simulation is given. The section starts with an explanation of the co-simulation mechanism (see Section 4.2.1). Afterwards, it presents different approaches how heterogeneous simulation tools can be coupled (see Section 4.2.2) and mechanisms, how the coupled tools can be synchronized (see Section 4.2.3).

4.2.1 The mechanism of co-simulation

Co-simulation frameworks are developed to cooperatively simulate complex systems by using specialized simulation tools. They can be realized by porting components from one simulator to another one which is time consuming and error prone due to simulator-complexity. The solution of this problem is to run each simulation tool in its own process and to connect them by the framework. [23]

Each tool has the responsibility on its own execution while the co-simulation framework realizes the synchronized progress of time and the information exchange. Thereby, the simulation tools do not communicate directly and maintain their own local state as well as an internal simulation time. Typically, a discrete event simulator is coupled with a continuous-time simulator. In that case, the simulation executes step-wise with a step granularity defined by the discrete tool. At the beginning of a synchronization step, both tools have the same simulation time [32]. A simple algorithm for synchronization can be described as follows.

The discrete tool uses the size of the time-step to set the simulation duration of the continuous tool by sending the value via the framework. The continuous tool advances its simulation time and stops either if the defined duration is reached or an event occurred. It notifies the discrete tool about the event and additionally sends the monitored variables as well as the internal simulation time. Afterwards, the discrete tool advances until both reach the same simulation time again [32].

However, the papers cited in this deliverable define different synchronization algorithms. These algorithms have various advantages with respect to the related use-cases.

4.2.2 Co-simulation in heterogeneous environments

A common application of co-simulation frameworks is the simulation of heterogeneous continuous-time and discrete event-driven simulations. Typically, continuous-time systems change their internal state continuously based on their inputs. The system dynamics are

represented by differential equations defining the transitions between the state variables. These equations are discretized and the time base is divided into small steps such that the system variables do not have a transition within a time step. [65]

This technique cannot be appropriately applied to discrete event-driven simulations since the time-step is difficult to select. A small step will waste simulation time if the states remain unchanged. Otherwise, using a long step, many events might be missed. In discrete event-driven simulation, the system time hops between events based on an event list. This list is managed by an event scheduler. The scheduler initializes the system state and the chronologically ordered list at the beginning of the simulation and selects the event on top at the correct instant. Afterwards, it wakes up the related process and adjusts the system time to the next event. [65]

Due to the different character of both simulation techniques, a heterogeneous environment requires a synchronization mechanism [65]. Hence, each simulator in a co-simulation runs its own process while a middleware carries messages between the tools and realizes time synchronization. Thereby, the middleware must ensure a consistent and on-time message delivery. A consistent message is delivered to the destination when its time-stamp is less than or equal to the current instant. Otherwise, it is out-of-order and cannot be processed anymore. On-time delivery requires the message delivery at the earliest next time step. [23]

4.2.3 Synchronization mechanisms between simulation tools

Ciraci et al. [23] present the FNCS framework as a federated approach to integrate power grid and communication network simulators. Providing a two-phase synchronization scheme, co-simulations involving transmission and distributed level grid simulators with a communication network simulator are supported. FNCS is implemented in C++ with interfaces for C, Java and Fortran. Although it does not implement the HLA standard [26], the framework follows the publish-subscribe design. This provides flexibility in integrated simulations.

The main components of the framework are the FNCS broker, the communication and time management components as well as the FCNS application component. The latter realizes an application layer running the communication network simulator. For each node, one component has to be created which forwards the messages. All simulators being federated by FNCS need to register with the broker which runs as a separated process. This behavior allows centralized control of the simulator processes. The communication between the broker and the simulator is realized by the inter-simulation communicator component which runs within the simulator processes. The communicator is managed by the communication management component. It provides interfaces for sending and receiving messages which can be used by extending the simulators. Re-use and flexibility are provided by identifying the nodes with names. FNCS automatically identifies the destination node by its name and delivers the message when it is safe. [23]

Finally, the synchronization scheme in [23] determines when to synchronize the simulators according to in-transit messages and request next time steps. One possible scheme is the conservative synchronization. It synchronizes the simulators at the smallest next time step of the power grid simulators. An extension enables the detection of packet loss to prevent unnecessary synchronization. Performance issues can be overcome using two different synchronization strategies. Those are *speculative failover* and *speculative recompute* [23]. The speculative failover makes speculative decisions about the instant of the synchronization. After a synchronization point, the parent process is duplicated. It continues executing using the conservative synchronization while the child speculates that there is no message exchange until the next synchronization point. If the speculation succeeds, the parent process is terminated. In case of many message transmissions, process management becomes costly. The speculative recompute technique allows child processes to run until they send a message and learn about the sending instant. This information can be used to speed up the simulation. Results show a performance improvement by 20% on average.

However, further analysis shows that the strategies are most successful when there is a large discrepancy in the simulators' time steps or when they do not send messages frequently.

Lai et al. [61] present a middleware with (i) interfaces to connect commercial power (PSCAD/EMTDC) and communication (OPNET) simulators, (ii) a synchronization method for the correct time and sequence of data exchange and (iii) a step adjustment algorithm balancing the requirements of accuracy and efficiency. The interfaces in the simulation software are provided by the software itself. Instead of using a continuous-time simulation, the power simulation is solved in a discrete manner. The events from both simulators are mixed according to their time-stamps preparing a uniform event-queue. When the simulation starts, only one process runs. An event from the communication system implies a breakpoint where the system suspends itself and starts to exchange information. Afterwards, the process restarts repeating the synchronization. Thereby it is possible to adjust the next synchronization point by including a command in the transmitted data. The algorithm uses the communication delay and error information contained in the current time step.

VPNET [64] integrates the Virtual Test Bed (VTB) and the OPNET simulation tool. The created co-simulation framework is capable of analyzing (i) how network performance affects the behavior, stability and safety of the controlled power system and (ii) how demands from the power system influence the design of the communication network. A co-simulation coordinator serves as a communication data channel between the simulation tools. It acts as a master managing the global communication step time and commands each simulator to take the next simulation step. To exchange data, both simulators provide interfaces for interaction. For example, the nodes in OPNET must contain an external Esys module to send and handle packets and an Esys interface to interact with any external C code. Each interface is bi-directional and multiple interfaces are possible. The reason is that each interface only supports one data type. In VPNET, the communication is realized using windows sockets for inter-process communication. Time synchronization between the continuous-time driven VTB and the event-driven OPNET is realized using a global time reference. This reference must run ahead of the local times and a sampling period decides when the coordinator reads data from VTB. The sampled value is sent to OPNET which simulates the network communication. During the delay, VTB executes with the previous feedback value until a new value is received. The simulation steps are synchronized by the coordinator which starts and stops the simulators.

The framework GECO proposed in [65] uses a global event-driven mechanism whose accuracy is tunable based on the time-scale requirements. GECO co-simulates continuous-time power systems (using PSLF) and event-driven communication networks (using NS2) realizing a heterogeneous environment. A simple synchronization mechanism would be an explicit time-stepped synchronization where several synchronization points are predefined. At these points, the simulations exchange information while they run independently in between. This method can easily cause simulation errors. The reason is that an interaction request appearing between the synchronization points can only be served at the next point. The error can be the same as one synchronization time step and might accumulate over time. However, it does not exist in real systems. The synchronization mechanism in GECO avoids these errors since the framework runs in a globally discrete event-driven manner. Each iteration round is treated as a special discrete event which are organized in a global event list according to their time-stamps. A global event scheduler realizes the global time reference and acts as a coordinator. Similar to the simple approach described above, the scheduler selects the next process with respect to the event list. However, the process can suspend itself after an event and yield the control back to the scheduler. In this way, each event can be processed immediately by the scheduler without any delay. By the coupling, the output event of the power simulation is the input event of the communication simulation and vice versa. As a mathematical formalism realizing the coupling, the Discrete Event System Specification (DEVS) is used. In the implementation, the global scheduler as well as the event list is derived from the NS2 simulation tool. A bi-directional C++ interface is

implemented between the two simulation tools to control the execution. Additionally, it is possible to integrate different simulation tools. However, this requires support from the tools and the coordination between them might be difficult.

4.3 Functional mock-up interface and High Level Architecture

The Functional Mockup Interface (see Section 4.3.1) and the High Level Architecture (see Section 4.3.2) can both be used to connect various simulation tools. Hence, the following two sections first present the standards as well as their (dis-) advantages. In the following section, possible combinations to overcome the disadvantages are illustrated.

4.3.1 The Functional mock-up Interface

The Functional Mockup Interface (FMI) is a tool-independent standard used to exchange dynamic models or to co-simulate them [15]. For this, it provides an interface which is implemented by more than 30 tools for version 1.0 and more than 25 tools for version 2.0. [8]

Tool-independent modeling languages aim at the ease of model exchange between different simulation tools. Many modeling languages provide low level interfaces for the exchange of models and their co-simulation but, in contrast to FMI, these interfaces are not standardized [15]. While there were different interfaces for both use-cases of FMI available in version 1.0, the main advantage of version 2.0 is the integration of both interfaces in one standard. Additionally, small details were improved and new features introduced. Hence, the usage of the standard is simplified and the performance is increased. [14]

FMI mainly consists of two parts, the interfaces for Model Exchange (ME) and for the co-simulation. The first interface aims at the creation of a modeling environment which is able to generate C-code of a dynamic system model. Another simulation environment is hence able to use the generated code as an input/output block. In general, models are described by differential, algebraic and discrete equations with time-, state- and step-events. The second type of interfaces is used for co-simulation. In such environments, two or more simulation tools (slaves) are coupled by a master algorithm. This algorithm is responsible for the synchronization of the simulation tools and the exchange of data between them at discrete communication points. In between, the subsystems are solved independently by their own solver. If the master algorithm connects models for model exchange, it also solves the models [14]. Although FMI supports different algorithms, it does not define one in the standard [25].

Components implementing FMI are called Functional Mockup Units (FMU). They consist of one zip-file which includes an XML-file, the model and, optional, additional data. The XML file contains the definition of all environmental variables as well as other information about the model. Capabilities of the slaves like the support of advanced master algorithms are included in a slave-specific XML-file. The model is deployed by a set of C-functions as source code or in binary for different platforms. Depending on the use-case, the code contains either the model represented by model equations (model exchange) or functions (i) to initiate a communication with a simulation tool, (ii) to compute a communication time step and (iii) to perform the data exchange. Further information may include documentation files or utilized libraries. [15]

In Cyber-Physical Systems (CPS), physical processes in an environment are controlled by embedded computers and networks [63]. Hence, these systems contain different domains like continuous-time and discrete event. Due to the different domains, there are various semantic gaps like untimed vs. timed semantics and signals vs. events [99]. There are several approaches bridging the semantic gaps between heterogeneous modeling formalisms. Tripakis [99] provides a complete methodology for modeling and simulation of heterogeneous systems. Those systems can be finite state machines like Mealy or Moore machines with their untimed semantics or a discrete event actor such as those used in Ptolemy. However, FMI uses a timed and persistent signal-based semantics. Tripakis models

different subsystems with various languages and tools. From each subsystem, he generates one FMU. Furthermore, he uses the results from [17] to co-simulate the subsystems. Müller et al. [73] embed FMI components for model exchange in the discrete event domain of the Ptolemy II simulation environment. This generic framework is used for the simulation of heterogeneous systems. It provides a discrete event and a continuous-time domain among others. Thus, many models could be described only in the simulation environment. Müller et al. [73] present the calculation of continuous results in advance to improve the simulation's performance. Additional advantages are the easier and faster model implementation as well as scalability of the approach. For this, they use the FMI++ library which is a high-level C++ utility package on top of the FMI ME specification. It provides several functionalities for handling FMUs which are not covered by FMI itself.

According to the authors of [27], CPS could benefit from a methodology which is capable of (i) rapid prototyping, (ii) the arrangement within well-defined hierarchies of possibly hybrid paradigms, (iii) distributed co-simulation in different (hardware) platforms and (iv) exploiting high performance computing resources. They target at the combination of different tools within a co-simulation framework and to enhance them by flexible prototyping capabilities of new components. For this, the authors suggest the Modelica language as a rapid prototyping platform which is assimilable to other tools using FMI. They address the impact of transferring Modelica into different classes of simulation tools such as general-purpose, domain-specific or academic research-oriented simulation environments. Examples are GridLab-D, TRNSYS and the HLA.

In safety-critical systems, deterministic execution is one important aspect to provide safety guarantees (see [56], Ch. 11.3.3). Broman et al. [17] focus on the deterministic execution of FMUs under the FMI standard. They implement a co-simulation where one FMU either contains its own internal simulation algorithm or serves as a gateway to a simulation tool. Deterministic co-simulation can be reached using optional capabilities from FMUs. If FMUs lack these functionalities, basic modeling capabilities like simple discrete event simulation or variable step-size numerical integration become unachievable. The provided extensions of the standard as well as policies for designing FMUs enable a master algorithm to query an FMU for the time of events being expected in the future. Furthermore, they show the possibility of deterministic execution if all FMUs are either memoryless or implement one of rollback or step-size prediction.

The master algorithm presented in [17] is exploited in [25] to reach deterministic execution. The authors provide an IDE called FIDE for building applications which use FMUs. FIDE is based on the Ptolemy II simulation environment and utilizes its user interface, its simulation environment as well as the code generation feature. The user interface facilitates the arrangement of a collection of FMUs into an ensemble. This ensemble is further compiled into a stand-alone application by an automated process. The approach bypasses the Ptolemy II execution environment supporting different master algorithms and possible extension to the API as defined by FMI. Using the master algorithm and the extensions by Broman et al., the co-simulation of continuous and discrete dynamics becomes possible. Preliminary results indicate a significant performance advantage of the author's master algorithm over the Ptolemy II execution engine.

A communication step size control algorithm is given by Schierz et al. [90]. They extend the results from [7] on reliable error estimation and communication step size control. Based on a mathematical analysis, the asymptotic behavior of the local error and two error estimates is studied. Both of them are candidates to adapt the communication step size automatically to the changing solution behavior during time integration. To analyze the local error, an error estimate is calculated and compared to user-defined tolerances. In case of coupled systems without algebraic loops, the global error is bounded by a multiple of the local error. The extrapolation of Richardson [85] is time-consuming but reliable. Schierz et al. extend the algorithm tailored to FMI CS which results in substantial savings of time. Using the local error estimate, a new communication step size can be calculated. Controlling the steps can

significantly improve the performance of the master algorithm, especially the computing time and the accuracy.

4.3.2 The High Level Architecture

The High Level Architecture (HLA) was developed by the U.S. Modeling and Simulation Coordination Office and has many similarities to FMI [10]. Using HLA, individual components (federates) can be connected to a set of interconnected components called federation. A federate may be a computer simulation, a supporting utility or an interface to a live partition [26].

Since the HLA was designed at a level independent of any languages and platforms, it can be considered as a protocol. Therefore, it supports solutions to the most common problems of interoperability [10]. All object representation is contained in the federates and the HLA imposes no constraints on what is represented or how, respectively. It only requires specified capabilities for the interconnection with other federates by the exchange of data.

Data exchange is realized using services implemented in the Runtime Infrastructure (RTI) which acts as a distributed operating system for the federations. To interconnect federates, the RTI provides basic services for data exchange or management of (i) objects, (ii) time or (iii) data distribution. Those services can be accessed via an interface supporting runtime interaction among federates and responding to requests from the RTI. The interface defines a functional way to access the services as well as an API. This API is available in different languages like C++, Java, Ada or the CORBA IDL.

Alongside interoperability, the HLA was designed for the reuse of the components. To reach this, each federate must document its object model using a standard Object Model Template (OMT). This model is intended for information sharing to facilitate the reuse [26]. Strassburger [96] lists four possibilities to make a simulator compliant with the RTI. Those are (i) to modify the simulator's implementation with appropriate extension, (ii) to add custom modules to the simulation tool, (iii) to use an external API defined as a, e.g., dynamic link library and (iv) to couple the tools by a proxy if they support facilities for external communication. Such communication mechanisms might be files, pipes or sockets.

4.3.3 Combinations of FMI and the HLA

There are different approaches combining the HLA and FMI. Awais et al. [10] propose to use the RTI for providing a generic and standalone master algorithm. The goal is usability such that components can be plugged and played on a variety of distributed environments. Using the RTI as a generic master provides time synchronization, mechanisms to access and modify a federate's state as well as the support of events. The latter is not fully supported in FMI 1.0 [10]. In the HLA, federates are capable of publishing data which is used by one or more subscribers. The description of the inputs and outputs is defined in a model-description file and the hosting code can perform the actions generically. For this, the description file contains directives declaring the variables' types or names among others. Since FMI does not support complex types as yet, all attributes of an FMI-federation can be accumulated in one class of the Federation Object Model (FOM) of the HLA. In this case, each FMU-federate creates a new instance of the main FOM class and gets updated if it subscribes to related attributes. Hence, an FMU-federate is an FMU which is compatible to the HLA. Ownership management separates the access to an attribute to only one federate. If the federate already ensures that kind of access, complex methods for the management are not necessary. FMI for co-simulation only partially supports events by using the function *StepFinished()*. Indeed, this function does not supply precise timing in FMI 1.0 while RTI provides this functionality by default. Their implementation of a simple application shows great opportunities of integrating both standards although there were some limitations in FMI 1.0.

In [11], Awais et al. show, how the strengths of the HLA and the FMI are utilizable for realizing a distributed heterogeneous simulation platform which supports different types of simulations in conjunction. Those may be fixed time-stepped simulations, continuous-time simulations or discrete event simulations. The authors present two kinds of FMU-federates, a fixed time-stepped as well as a discrete event-based. The first type progresses in fixed time-steps typically using the HLA “time advance request” and “time advance request available” services. A discrete event-based FMU-federate proceeds based on internal and external events. This kind uses the HLA “next event request” as well as “next event request available” services. The process of time based on events requires to look in the future and to predict the instant of the next event. Both types of federates are generic unless there are limitations by the simulation tools. To support the requirements of both types, they present two different algorithms. These algorithms are based on the idea of zero lookahead based simulations. A lookahead is the minimum amount of time within which a federate will not generate any new event. The first algorithm is a fixed time-stepped algorithm. In a heterogeneous environment, the fixed step size may be a problem since there might be loss of information. One solution is the usage of episodic simulation in which the lookahead can be zero and federates may have different time steps. This is possible due to the HLA “time advance request” and “time advance request available” services. However, in case of variable step sizes, the “next event request available” service should be used instead. If the federates demand equal time advances, they stay in synch and have the possibility to exchange data at the end of an episode. It is guaranteed that updates created from a federate running on a slow machine will be delivered before the end of the period. The discrete event algorithm uses a function to predict the instant of the next event. For this, the function analyzes the output produced by the models. Its significant advantage is the reduction of the number of events compared to the fixed step size algorithm.

While the papers previously mentioned only refer to using HLA as a master for FMI, Garro and Falcone [34] investigate how to combine both standards from two perspectives. These are on the one hand HLA for FMI (i), on the other one FMI for HLA (ii). To support the first case, some extensions to FMI to include HLA are presented. The model description XML file is extended using tags which cover specific HLA functionalities. Those are, e.g., settings for time, declaration or ownership management among others. Additionally, a C/C++ shared library is created which defines related HLA commands. Hence, the model defines the FMU’s simulation logic according to FMI and uses the HLA features to make the FMU compatible with the HLA-based simulation environment. Advantages of this approach are (i) greater integration and execution control of FMUs in a HLA federation, (ii) reduced development cost and time since an FMU becomes reusable and (iii) better performance due to distributed parallel execution of FMUs sharing the same HLA simulation platform. However, this solution requires an extension of the FMI standard as well as adding capabilities to the FMU solvers for interpreting and executing HLA-based operations. With respect to the second case (FMI for HLA), two approaches based on well-known design-patterns are proposed. Those approaches are adapter- or mediator-based. The adapter is composed of two elements. The first one is an FMU containing the component’s behavior and its solver while the second component is the adapter. It manages the communication between the FMU and the RTI as well as the FMU’s lifecycle. The adapter is mainly a design pattern which connects the interfaces of two components without changing the components’ source code. This software layer interprets the application specific services and provides capabilities for communication and monitoring. However, the structure of the mediator differs from the adapter. It consists of a set of FMUs, each defining a dynamic model, and an HLA federate. The latter contains its own simulation logic and uses the FMUs to simulate specific components. In fact, the mediator is a software layer encapsulating the modalities for the interaction between the FMUs and the HLA federate. Since the FMUs and the federate can only communicate via the mediator, dependencies between them are reduced and coupling is lowered.

Neema et al. [75] identify challenges of integrating FMUs for co-simulation (CS) in general and via the HLA presenting a novel model-based approach. This approach can rapidly

synthesize an effective integration of available FMU-CS components with specialized simulation packages. The authors target on the co-simulation of CPS using different solvers and step sizes to overcome significant non-linearity and discontinuities. If the system is split at the wrong place, inefficiencies due to time management, composition restriction, data exchange and stability issues may be introduced. The problem of distributed simulation is solved using the HLA. Its key benefit is its distributed discrete event model of computation which allows flexibility to use any internal solver and model of computation. However, the HLA does not formalize any methods for developing interactions and models used by the federates. Furthermore it does not provide mechanisms to easily move simulations between the nodes. Additionally, the HLA needs a significant amount of hand-developed integration code which is tedious and error-prone. Using the HLA as a master for co-simulation of FMUs provides flexibility to exploit other simulation types. Neema et al. develop a platform which addresses important challenges like modeling interactions and shared objects, data exchange mechanisms and modeling the deployment of distributed simulation tools. Additionally, the platform provides a standard master algorithm for FMI co-simulation, multi-rate modeling with dynamic time-management as well as a toolset to generate necessary objects for rapid synthesis of simulations. The approach is based on the author's Command and Control Wind Tunnel (C2WT) [42] and automatically wraps FMUs to HLA federates. Those can be simulated with different clocks while time management is realized by the HLA.

C2WT is a distributed simulation environment for large-scale command and control systems. It enables the interaction and communication of different simulation engines and it relies on the services of the HLA and its RTI. Developing the integration code for the HLA as well as testing and deploying various run-time components across multiple platform-specific simulation tools is highly challenging. C2WT provides a holistic modeling and management environment using a custom Domain-Specific Modeling Language (DSML) implemented in the Generic Model Environment (GME) [75]. DSML is able to capture integration information about the model's configuration and its role in the environment. For this, the language provides all the primitives required to specify the integration, deployment and execution of the simulation [42]. The platform is additionally related to a suite of model interpreters for the coordination between the integration model and the platform-specific simulation tools. It facilitates the rapid development of models and their usage throughout the simulation environment's lifecycle by automatically generating the integration code. Beyond, C2WT supports real-time and as-fast-as-possible simulation execution. However, the current real-time simulation requires an execution which is faster than real-time. Human interaction can be performed using HLA-interaction mappings. Furthermore, the platform supports extensive experimentation, message and state variable logging as well as analysis functionalities [75].

4.4 Internet-based co-simulation

In the next sections, different approaches for Internet-based co-simulation are presented. Those are mainly settled in the research-area of electric power grids, but also in network control systems and networked multi-core chips. First, techniques for coupling tools via TCP/IP-sockets are shown in section 4.4.1. In section 4.4.2, approaches for distributed co-simulation follow.

4.4.1 Co-simulation via TCP/IP sockets

Today, the TCP/IP-stack is mainly used for communication via the Internet. In this section, different approaches for co-simulation using the stack to communicate are presented.

Harding et al. [41] introduce an interface between MATLAB and OPNET to allow the simulation of Wireless Networked Control Systems (WNCS) with Mobile Ad Hoc Networks (MANET). In this use-case, the WNCS is simulated using MATLAB while OPNET simulates the network. Thereby, there are two instances of MATLAB used, one for a controller and another one for a plant. OPNET uses threaded sockets to wait for messages which are forwarded to the tool via the Esys API. The related sockets in MATLAB use Java sockets

instead. A value to be sent is encapsulated into a packet and sent to the controller processes in OPNET and vice versa. The arrival of a message to a process in OPNET is signaled by an Esys interrupt caused by the co-simulation.

A co-simulation platform combining OPAL-RT for power and OPNET for communication system simulation is presented in [13]. This platform supports simultaneous simulations of power and communication systems in real-time but also in non real-time. In case of the latter, two simulators run separately but synchronization is not necessary. One tool runs until it finishes and stores its results in a file which is used as input for the other tool afterwards. Only the simulation time must be synchronized to realize accurate information exchange between the simulations. Due to the asynchronous operation of two simulators, the accuracy of the results might be limited. In contrast, real-time co-simulation requires an interface for information exchange and synchronization. The proposed platform exploits the interface as a data buffer which allows real-time packet exchanges using protocols like TCP or UDP. A use-case to test the platform is placed in the context of a distributed automation application. In it, the nodes are connected by Ethernet and Wi-Fi.

The approach illustrated in [78] deals with networked multi-core chips in contrast to the previous platforms that concentrate on the simulation of distributed power grids. In such systems, multiple multi-core chips based on a network-on-chip are interconnected building up a distributed system. Such systems are co-simulated using Gem5 for the chip-level and OPNET for the cluster-level. For the connection and the synchronization of both discrete event simulators, local simulation controllers are introduced as additional components. The usage of TCP/IP sockets enables the execution on multiple physical machines. In this environment, the local controllers determine the satisfaction of dependencies to perform a simulation step. The execution of the simulation tools is based on its own local event calendar. Additionally, a global calendar contains events that have causal relationships in the co-simulation. The calendar represents the execution order's time model and is used to synchronize the simulation tools. Thereby, each event in the local calendar must be managed and modified using control messages. To perform the communication, the local controllers provide interfaces from and to the network and introduce basic gateway functionalities. Examples of such functionalities are queuing and mapping of messages. Furthermore they are responsible for converting the message formats and for mapping the incoming message's destination address to the target application. Synchronization is realized by a socket-based interleaving approach. It exchanges data and provides access for mutual modification of local calendars. At one time, only one simulation tool is running mutually exclusive based on both local calendars. If one tool requires sending a data message, the tool is suspended and an interrupt is sent to the other tool signaling the communication request. As soon as an answer arrived, the simulation can continue.

4.4.2 Distributed co-simulation frameworks

Besides a parallel execution of simulators in geographically distributed machines connected via LAN or WAN (Wide Area Network), distributed co-simulation has further advantages. It allows (i) project decentralization, (ii) design and validation of a system by geographically distributed teams, (iii) intellectual property management where a component is simulated without publishing its description, (iv) simulator's license management and (v) resource sharing. Although parallel resources are usable, there is network communication overhead which might increase the co-simulation execution time. Using a local simulation does not introduce a communication overhead. Indeed, it does not present the benefits of a distributed co-simulation. [5]

Amory et al. [5] depict a hardware and software co-simulation tool composed by geographically distributed computers which communicate via a co-simulation backplane. The backplane reads a file describing how the tools are connected, launches the simulators and controls the simulation process. The authors present a heterogeneous approach that can be used on different system levels like architecture-level or cycle-level. Their co-simulation tool

validates each module of the design according to its language and manages communication between the simulators. While the co-simulation is evaluated in terms of CPU time for finishing in a distributed system, the paper additionally presents an evaluation of the CPU time used for simulation and communication. In the environment, the user must add the appropriate communication library to each module. Software modules are described in C using ComLibC whereas VHDL (Very High Speed Integrated Circuit Hardware Description Language) is used for hardware components (ComLibVHDL). As an interface between VHDL and the C code, FLI can be used which is an extension of VHDL. The communication library encapsulates the communication primitives to the co-simulation backplane and realizes the communication using UNIX sockets. However, coordinating the communication is the backplane's main functionality. It is independent from simulators and new languages can easily be integrated. A coordination file describes the modules being instantiated with related information like name, host, simulator, etc., and the connections between them. In contrast, the lack of portability of the C/VHDL interface is one weakness in addition to a missing global synchronization mechanism. Furthermore, the communication between in the co-simulation introduces an overhead as shown by the simulation results.

Another co-simulation environment for electric power grids and communication networks is the Federated Simulation Toolkit (FSKIT) [54]. It couples the GridDyn simulation tool (power) with the ns-3 network simulator. FSKIT is a high-performance dynamic simulation platform supporting large-scale integrated transmission, distribution and communication systems. The platform acts as a middleware providing time control on the one and facilitating communication between federates on the other hand. A simple and lightweight design enables the coupling of multiple continuous-time and discrete event parallel simulations. Maintaining the correct temporal order of events is realized by three different approaches, a conservative, a fixed time step and an optimistic one. The first approach synchronizes all events which maintains the correct order but introduces many synchronization points. A fixed time step approach synchronizes the events in regular intervals. On the one hand this behavior improves execution effectiveness and parallelism, on the other one it may introduce delays in the event delivery. At last, the optimistic approach uses speculative execution maintaining accuracy. However, the requirement of roll-back capabilities of the simulators can have a significant impact on their implementation. FSKIT provides an asynchronous API which allows overlapping communication and execution. Since FSKIT is designed for high performance platforms, it uses the Message Passing Interface (MPI) for inter-process communication. Besides high performance, MPI also enables the communication between processes on distributed hosts. To be coupled with FSKIT, a simulation tool is wrapped in a class inheriting from a FederatedSimulator interface. This interface contains the required functions to advance time and the resulting class is registered with FSKIT. The registration is also performed to communicate which is realized using LogicalProcess objects. Therefore, FSKIT provides a parallel scheduler that controls the parallel time advancement on the one and functions for sending an event message on the other hand.

The electric power and communication synchronizing simulator (EPOCHS) [43] is a distributed simulation environment for electric power and communication co-simulation. It integrates multiple research and commercial off-the-shelf systems using different simulation tools. Those tools are the PSCAD/EMTDC electromagnetic and the PSLF electromechanical transient simulators as well as NS2 for network simulations. EPOCHS is based on the HLA and agents. Hence, the simulation engines can be federated only using their built-in APIs and the framework hides the complexity in combining simulation systems. AgentHQ is a unified environment for agents which act like proxies. It provides functions to send and receive messages and is a discrete event system. All events are processed as they occur and they are routed to the affected agents. The events' temporal order is maintained using a time-stepped model. All federates simulate until a defined simulation time is reached and synchronize themselves via the RTI. Before the simulations continue execution, the RTI selects the next synchronization point based on fixed time-step lengths. Those lengths are user-selectable for each step. Sending the messages between the simulators is controlled by

AgentHQ in a round-robin manner. However, it is also possible that the discrete event simulator NS2 receives a communication request between two synchronization points which requires interaction with the power simulation. In this case, the agent queues the message until the next synchronization point occurs. Indeed, this mechanism may lead to errors as explained above.

4.5 Security mechanisms applicable for secure communication in distributed validation frameworks

This section deals with the cryptographic security mechanisms and the related techniques in the field of symmetric and asymmetric encryption necessary for a secure communication. Since data integrity plays a major role in the assurance of the accuracy of communication data transferred within a channel, the focus especially lies on techniques for the assurance of data integrity and authentication.

4.5.1 Data Integrity and Cryptographic Hash Functions

Data integrity assures and maintains the accuracy of data and aims on preventing unintentional changes to information. Furthermore, it mainly affects reliability, safety, confidentiality and integrity attributes of a safe and secure system. Problems in that domain comprise unintended changes to data due to storage, retrieval or processing operations. This also includes targeted changes, unexpected hardware failures, human errors and malicious attackers. Measures to preserve the integrity of data are diverse, including the application of hash functions as well as checksums and cyclic redundancy checks.

Integrity protection may be based on symmetric authentication techniques. In that case, the same secret key is required for generating as well as verifying a Message Authentication Code (MAC). The MAC is a cryptographic checksum that authenticates a message. In general, hash functions or symmetric encryption algorithms are used to generate MAC values. The number of secret keys required to be distributed through a secure channel grows exponentially with the number of network components. The reason is that each pair of components needs to agree on an individual key.

Cryptographic hash functions play a fundamental role in data integrity and message authentication. A hash function creates a digital fingerprint of an input string by mapping bit-strings of arbitrary finite length to a string of fixed length. Message authentication codes represent a distinct class of hash functions which require the secret key besides the input string (the message, respectively) as the functional input. MACs are designed to produce a fixed-size output independent of the input string's length. This output is practically infeasible to reproduce without the corresponding symmetric secret key. A MAC value is calculated either using a hash function (Keyed-Hash MAC (HMAC)) or a symmetric cipher (CMAC). As only the knowledge of the secret key permits the generation and verification of the MAC, this key must be shared with all possible validators.

4.5.2 Private-Key Cryptography

The private-key cryptography, also known as symmetric encryption, uses the identical key for both encryption of messages and decryption of ciphertexts. Based on the short encryption keys compared to those of the asymmetric encryption, the encryption speed is much faster.

However, the main drawback of symmetric encryption relies on the key distribution since there is only one key for the en- and decryption. In order to establish a secure communication based on symmetric-key algorithms, the key has to be distributed securely over all participating entities. A further problem is the damage on securely stored data when the secret key will be compromised. Since the key is shared with the opposite entity for a secure two-way communication, an attacker could get hands on ciphers and decrypt all of them with the gathered secret key. Therefore, a new key has to be generated and newly distributed in case of a disclosure of the secret key.

The most common and widely used symmetric-key mechanisms are AES (Advanced Encryption Standard) and 3DES (Triple Data Encryption Standard).

4.5.3 Public-Key Cryptography

Public-Key Cryptography (PKC), also known as asymmetric encryption, represents a cryptographic system based on key pairs. Those key pairs represent the main difference in comparison to the private-key cryptography. On the one hand, an asymmetric cryptographic system contains a private key (also referred to as a secret key) which belongs to the owner and is only known by him. On the other hand, it includes at least one public key. This key may be publicly distributed in order to allow entities to operate with the cryptographic system. While the public key can be only used to encrypt messages, the private key is utilized to decrypt the received messages encrypted by the public key.

Public-key cryptosystems represent a fundamental security component of various Internet standards, such as the TLS (Transport Layer Security) protocol. There are various types of public-key algorithms known and certain PKC algorithms fulfill single functionality. While some algorithms provide only a key distribution and secrecy function, others offer only digital signatures. Nevertheless, there are commonly used PKC algorithms which are able to provide both functionalities. Basically, there are various asymmetric encryption techniques known. Since digital signatures represent an important component for data integrity and authenticity, the subsequent focus lies only on digital signature-providing algorithms. Those are RSA, Digital Signature Algorithm (DSA), Elliptic Curve DSA and Edwards-curve DSA.

4.6 Research gap for state-of-the-art

One goal of the distributed simulation and validation framework is the connection of various simulation tools and even physical devices in a discrete event co-simulation over the Internet. To reach this goal, the initial requirements and design goals described in section 4.1 need to be fulfilled. Important aspects are scalability and generic interfaces to the simulation tools as well as a mechanism for data exchange and synchronization.

Most of the frameworks for Internet-based co-simulation described in section 4.4 provide a mechanism for synchronization and data exchange in distributed co-simulation over the Internet. They couple the continuous-time domain needed for simulated plants with the discrete event domain required for the simulation of networks and control applications realizing discrete event simulations. Tools like FSKIT or FNCS are designed to support efficiency and high performance while the framework described in [61] supports an adaptable communication step-size.

However, the frameworks all suffer from main disadvantages. They are all designed for specific simulation tools without any generic interfaces or compatibility to existing standards. Determinism and reproducibility as HIL/SIL co-simulation are both unaddressed as well as distributed event calendars. Furthermore, the aspect of applicability is not considered. Although the approaches provide the possibility to re-use simulations designed for the coupled tools, modularity of components or to plug and play them are unattended. Another lack concerns the requirements for configuration. Except the framework described in [78], none of the other authors points out whether their tools are configurable or not.

Both, the FMI and the HLA can be used to overcome the lacks of adaptability and interoperability. The standards are designed to connect and re-use existing models of various existing simulations as long as they are compatible. While FMI provides generic interfaces to the simulation tools, it suffers from a master algorithm which synchronizes the simulations and provides a mechanism for data exchange. However, the combination of FMI with the HLA solves the problem. There are different approaches described in section 4.3.3 where FMI exploits the RTI of the HLA to act as a master. These approaches realize a distributed discrete event simulation where different domains are coupled. Other references provide solutions to address most of the remaining requirements. For example, [25] and [17]

both address determinism and reproducibility of simulations, C2WT [42] provides a heterogeneous, scalable platform and [11] as well as [90] present an algorithm for adaptable communication step-size.

Although many requirements and design goals for the distributed co-simulation framework are covered, there are still unsolved problems. The approaches do not use distributed event calendars or support remote procedure calls. Furthermore, there are the same lacks for configuration requirements like in the Internet-based co-simulation frameworks. With respect to timing requirements, [75] supports different time-resolutions of the coupled simulations while the C2WT framework manages events with the correct temporal order. Indeed, the other authors do not address any timing requirements.

The applicable security mechanisms and their algorithm parameters strongly depend on available hardware resources of the participating entity, mainly on the present computational power. Therefore, dedicated cores for cryptographic calculations as well as for secure key storage are often not considered for embedded systems. A further security gap in the domain of low computational power entities is the physical location and accessibility. Those entities are more vulnerable against malicious attacks and therefore represent a major threat to the overall secure environment.

Due to their properties and the coverage of many requirements defined for the distributed co-simulation framework, the combination of FMI and the HLA is a promising approach. Therefore, the research gap contains two parts. On the one hand, the different extensions of FMI have to be combined to address the remaining requirements for co-simulations. On the other hand, further solutions are required. Those solutions are related to the requirements for distributed co-simulation, timing, configuration and security of the framework.

Chapter 5 Safety

5.1 Initial Safety Requirements

The distributed simulation and validation framework is a tool to test a safety application. Assuming this tool “cannot introduce errors, but may fail to detect them” [101] and the principle that testing cannot prove the absence of errors - the derived safety requirement is:

Distributed simulation and validation framework – error detection capability

> *The distributed simulation and validation framework shall be able to detect all types of errors in a safety application that are relevant to the used test level.*

The tool’s error detection capability is sufficient if it fully supports the needs of the safety application validation (see chap 5.1.1) and is provided with adequate quality (see chap. 5.1.2).

5.1.1 Support for safety application validation

In order to support the safety application validation, the distributed simulation and validation framework must meet the expectations of the surrounding processes using it. Also the application safety functions to be validated and the safety related tool chain interfaces have to be satisfied. The tool must support the configuration and evaluation of its behavior in terms of real world characteristics (e.g. timing etc.). Chapter 6 and Chapter 7 of the present document contain more information about how the framework will support the safety application validation.

5.1.2 Qualification of the distributed simulation and validation framework

For using the distributed simulation and validation framework as a tool in the development process of a safety system, a tool qualification shall be performed. The goal is to generate evidences that it is free of faults which avoid the detection of errors in the developed safety system. For an adequate tool qualification of the distributed simulation and validation framework, the below mentioned tasks have to be performed.

Tool Risk Assessment

A risk assessment of the intended tool usage and relations to surrounding processes is needed to allow the evaluation of the safety related impacts caused by possible tool failures.

Tool Qualification Plan

State the tool classification according to the applicable standard EN 50128 [21] to mitigate safety related impacts caused by possible tool failures. The identified tool configuration, tool activities in the development process and tool qualification activities are described.

Tool Operational Requirements

Specify tool operational requirements derived from surrounding process expectations and interfaces to the tool chain. Stakeholder input shall be used for requirements that cover expected functional behavior, expected indication of abnormal activation modes as well as inconsistent input and the operational environment (operating system and hardware). The requirements shall also clarify information necessary for the tool’s installation as well as operation and which software development processes are performed by the tool.

Tool Qualification Records

Record the results of the tool qualification activities (for example, test cases, procedures, and results).

Tool Configuration / Quality / Change Management Records

Record the configuration- (e.g. configuration identification lists, baseline, etc.), quality- (e.g. QA review or audit reports, meeting minutes, etc.) and change- (e.g. Problem reports, change requests, etc.) management process activities.

Tool Quality/ Configuration/ Change Management Audit

Perform a quality, configuration and change management audit.

Tool Qualification Accomplishment Summary

Create a summary about tool status (Intended usage, integrated toolset functionality, fault handling, configuration, maintenance, identified shortcomings) about the compliance with the tool qualification planning.

5.2 State-of-the-art

The state-of-the-art safety aspects regarding the distributed simulation and validation framework are not specific to its function but to its nature as a tool used in the development process of a safety system. The expectation from such a tool is that it is free of faults that prevent the detection of errors of the safety system. A minimum quality of the tool must be demonstrated to get evidence for the claimed freeness of faults. The process and method to generate such evidence is commonly called tool qualification.

Tool qualification is a long time research and standardization issue in the aircraft industry. The following analysis results are based on one output “Tool Qualification in Multiple Domains: Status and Perspectives” [48] of these activities that covers the status up to 2014.

5.2.1 Tool Categorization

All standards categorize tools based on the potential impact of a tool failure in combination with additional measures provided by the safety application development process. ED-12 (DO-178) [29], ED-215 (DO-330) [30] and ISO 26262 [47] currently have the most elaborated categorization.

The tool failure impact can be summarized by:

1. Tool failure does not impact the executable code (including data) of the safety application directly or indirectly.
2. Tool failure impacts the test or verification of the design or executable code. Errors in the tool can fail to reveal defects but cannot directly create errors in the executable software.
3. Tool failure impacts the executable code of the safety application directly or indirectly.

Additional measures to prevent or detect tool malfunctioning may be manual checks of the tool output or divers tool chains among others.

The tool failure impact defines the rigor of the required tool qualification. This definition is combined with measures that prevent the tool from malfunctioning or detect that the tool has malfunctioned. The rigor of tool qualification - the tool category - is defined by the tool classes T1..3 (EN 50128, EN 61508), the tool qualification levels TQL-1..5 (ED-12, ED-215) or by the tool confidence levels TCL1..3 (ISO 26262).

5.2.2 Requirements on Tool Qualification

All standards put requirements on the tools qualification based on their tool category. ED-215 (DO-330) currently has the most elaborated requirements.

An overview of common tool qualification requirements with basic rigor requirements (justification or evidence) related to tool category (T1, T2, T3) is shown in Table 5-1.

Table 5-1 Common Tool Qualification Requirements

Common Tool Qualification Requirements	**T1**	**T2**	**T3**
Compliance to Intended Usage.			
The intended **usage** and the resulting **tool operational requirements (TOR)** shall be specified: - Usage Application based TOR - Operational Environment - Safety Related System - SW Application Needs/ SWSIL - Tools´ Life Time/ Availability SW Requirements/ Arch. based TOR - SW Application Development Method/ Process Requirements (e.g. support of design/ programming error detection, design method) - Standards Compliance (e.g. Compilers, Design Tools) Integrated Toolset based TOR - Integrated Toolset Needs - Integrated Toolset Cooperation Requirements …	Evidence	Evidence	Evidence
The classification of the tool shall be justified. Combined tools (development & verification) shall be qualified like development tools (T3) unless partitioning can be shown.	Justification	Justification	Justification
The tool (as part of a set of techniques, methods and tools) shall satisfy the tool operational requirements as set by the software requirements at the required SIL. Requirements-based coverage to the TOR shall be analyzed.		Justification	Justification

Common Tool Qualification Requirements	T1	T2	T3
The tool shall satisfy the tool operational requirements as set by the needs of the application, and of the safety related system. Requirements-based coverage to the TOR shall be analyzed.	Justification	Justification	Justification
The availability of the tool over the whole lifetime of the software shall be considered.	Justification	Justification	Justification
Compliance to Integrated Toolset			
The tool shall satisfy the tool operational requirements as set by the needs of the integrated toolset.	Justification	Justification	Justification
The tool shall work co-operatively such that the output from the tool shall have suitable content and format for automatic input to a subsequent tool.	Justification	Evidence	Evidence
Correct Functionality			
The tool shall have a specification or manual which clearly defines the behavior of the tool and any instructions or constraints on its use.		Evidence	Evidence
Evidence shall be available that the output of the tool conforms to the specification of the output. (Documentation of all analysis/ validation steps shall be available) - A suitable combination of history of successful use in similar environments and for similar applications - *or* - Tool validation e.g. as a comparison of the tool outputs to the output of the manual process that is replaced by the tool. - *or* - Compliance with the safety integrity levels derived from the risk analysis of the process and procedures including the tools. (Tool developed to EN 50128)			Evidence
Fault Handling			
Potential failures which can be injected into the tools output (due to e.g. systematic tool faults, random tool faults because of abnormal operating conditions,…) shall be identified.		Evidence	Evidence

Common Tool Qualification Requirements	T1	T2	T3
There shall be measures that avoid or handle such potential failures, e.g.: - Measures within the executable safety related software. - *and/or* - Other appropriate methods for avoiding or handling failures introduced by tools (e.g. manual check, diverse tool chains, …).		Evidence	Evidence
If there is no evidence for correct functionality available, there shall be effective measures to control failures of the executable safety related software which result from faults that are attributable to the tool, e.g. - Diverse redundant code which allows the detection and control of failures resulting in faults introduced by a tool.			Evidence
Configuration Management			
Configuration management shall ensure that the tool used - is of a justified version - *and* - is compatible to the versions of the rest of the toolset		Evidence	Evidence
Maintenance			
Justification for a new version of a tool may rely on evidence provided by an earlier version if sufficient evidence is provided that: - the functional differences (if any) will not affect tool compatibility with the rest of the toolset, - *and* - the new version is unlikely to contain significant new, unknown faults.		Evidence	Evidence
Identified Shortcomings			
Any additional measures which address any identified shortcomings shall be justified and evaluated.			Evidence

5.2.3 Tool Usage Requirements

All standards require some user documentation. Only ED-215 (DO-330) formally identifies a set of objectives with related activities and tool qualification data to be satisfied by each tool user.

Regarding the user skills, EN 50128, ISO 26262 and EN 61508 [20] put requirements on skills of software team members, including for tool users.

The summarized conclusion is to apply the latest standard of ED-215 (DO-330), with respects to the required personal skills stated by EN 50128 / EN 61508, on the development of the distributed simulation and validation framework.

5.3 Research gap for state-of-the-art

The Safe4RAIL project is dedicated to the railway domain. Hence, the proposed simulation and validation framework will be used in the railway domain safety application validation processes. For this, it requires the fulfillment of the EN 50128 requirements regarding tool qualification.

As explained in chapter 5.2, the analysis of the different standards regarding tool qualification presents the conclusion that nowadays the standard ED-215 (DO-330) is more advanced than EN 50128 in certain aspects of the tool qualification process. ED-215 (DO-330) was developed from ED-12 (DO-178), Software Considerations in Airborne Systems and Equipment Certification. However, this standard was developed as a new domain-independent document, with the idea of being able to apply in domains different than avionics. Hence, it will be possible to apply DO-330 / ED-215 thanks to its domain-independent nature.

Safe4RAIL will analyze tool qualification requirements given by EN 50128 and DO-330 / ED-215. The project will elaborate a concept and requirements for the tool qualification of the proposed simulation and validation framework. During the development of the framework, the suitability of the concept and the satisfiability of the tool qualification requirements will be evaluated. Tool qualification solutions related to detected problems will be analyzed for general applicability. A full tool qualification will not be performed during the Safe4RAIL project. However, the evaluation results will be used create to a consolidated set of tool qualification requirements and a tool qualification plan.

Chapter 6 Test operation and test automation

6.1 Design goals for test operation and test automation

The specific requirements shall not be repeated here and can be read at ASAM - Association for Standardisation of Automation and Measuring Systems [33]. The following section cover requirements for the connection of the test automation (see Section 6.1.1), those solved by the test tool (see Section 6.1.2) and execution steps for test automation (see Section 6.1.3).

6.1.1 Design goals for the connection of the test automation

As an overview, the design goals for the connection of the test automation to the system under test can be denoted as follows.

Communication of simulation components

Must be separated into the model of computation as well as the simulation architecture description and implementation.

Model of computation

The HILSF must allow all relevant models of computation required for the test use to be executed. In case of a distributed simulation, the HILSF must ensure the same model of computation as well as the simulation's remote instantiation.

Simulation architecture description and implementation

The HILSF must provide a modeling language for describing of the simulation architecture and the simulation hardware executing the models.

Simulation control

The HILSF must allow exact setting and supervision of the following properties of each simulation members. Those properties are the local simulation time, the liveliness of the execution and the status of the simulation instantiation steps.

Simulation operation

In addition to the above-mentioned requirements, the test automation driving the simulation must implement a test automation and virtualization interface as well as signal injection and measurement mechanisms.

6.1.2 Design goals solved by the test tool framework

Beyond the requirements described in Section 6.1.1 on the HILSF, the industrial success of any such solution will depend on more requirements solved by the test tool framework around the simulation at its core. Therefore, these requirements shall be mentioned here but are beyond the focus of Safe4RAIL project.

Modelling and model management

Models representing the controller are easily mastered, but the availability of a plant model with correct behavior and appropriate simulation performance is a constant challenge. Moreover, the re-use of plant models is crucial for reasons of cost and quality. The delivery chain of models to the SIL / HIL user is a core competence and not standardized, but equally complex as the integration planning of the final vehicle.

Mastering Simulation

A simulation is an abstraction. Judging the validity of a simulation w.r.t. the test executed using the model is crucial for acceptance of the simulation result. It requires tools and methods which are not part of the HILSF standardization.

Test script automation (following Evoke Technologies [98])

The test script automation handles scripts and data separately. It creates libraries while following coding standards and offers high extensibility. Furthermore, it provides less maintenance and script/framework version control.

6.1.3 Execution steps for test automation

Testing comprises as multitude of actions, implemented as functions integrated into a test framework. Only a subset of these functions is inseparately related to test operation and automation. The steps to the execution of an automatic test and required to be supported by the HILSF shall be explained in the following.

Configure

This step models the configuration of the simulation. It contains connections of the sub-models, the model of calculation for the integrated system, the timing of the models and their connections and the allocation to simulation nodes.

Build

The communication according to the configuration is implemented. This can be an integral part of the next step “instantiate” in case of configuring an exisiting simulation communication backplane or it can be a separate step in case of the calibration.

Calibration

The parameters of each simulation must be calibrated at build time or at the init phase of the first simulation execution. In railway systems, calibration is included at build time due to legal reason.

Instantiate

The HILSF must provide an API to be called for instantiation of all parts of a simulation.

Execute, stimulate, record and supervise

The HILSF must provide the function of starting all simulation. Moreover, it must provide APIs to (i) allow sending stimulus signals from the test automation to each simulation, (ii) allow sending signals from each simulation to the test automation and (iii) for assessing the liveliness of each simulation.

Close

The HILSF must provide an API for reversing the instantiation of the simulations.

Analysis and assessment

Includes the comparison of the recorded behavior with the expected behavior.

Report

Generates a report on the results.

6.2 State-of-the-art of test operation and test automation

The state-of-the-art is represented by COTS tools successfully implementing all requirements of the preceding chapter, some of them mentioned in Figure 6.1. The challenges in test operation and automation follow the today's challenges for system development. Some important issues are denoted in the following.

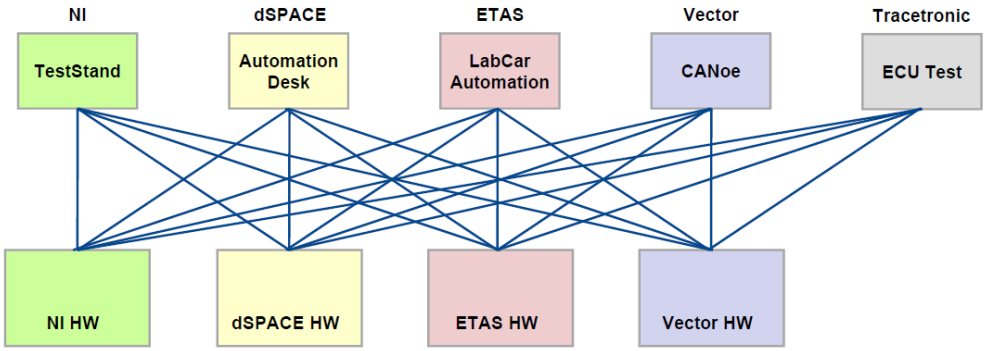

Figure 6.1 Desired interoperability of test automation tools and test execution platforms [from 82]

Distributed development

As already described above, the exchange or connection of simulations into one system simulation is a challenge. For the testing, this entails the integration or the exchange of test sequences among companies. This is addressed by defining universal or domain specific standards for test languages, e.g. [89].

Distributed function

The implementation of functions in a distributed manner across different controllers imposes difficulties in accessing and assessing the remote behavior. In a laboratory, this can be achieved by using distributed test frameworks as HILSF. In the real implementation, the need for such tests occurs typically after commissioning or maintenance. Hence, testing functions are implemented in the series software. An efficient tool chain for transforming the desired test sequences from test lab to series software is a challenge under development in the automotive industry.

Model based development

Model based development exists for nearly every test language. It is not standardized and as varying as available modeling tools for system development. In terms of such complex, distributed test environments as Safe4RAIL is intending to create, a usable model based test framework configuration and operation is desirable. However, it is not yet state-of-the-art of any market-available tool.

Top down development and test specification vs. refactoring

In established industries like automotive or railway, most systems are not built from scratch but based on legacy components. For them, the necessary abstract models for a model based test specification do not exist. As refactoring is a major challenge for the system development the efficient extraction of the models for test specification is the corresponding, unsolved challenge for the test engineer.

6.3 Research gaps for state-of-the-art

At present, there is no major research gap in terms of test automation in the aspect of executing a system under test and with the focus of Safe4RAIL as shown in chapter 6.1.

However, successful test operation is also limited by matching the models with the test use cases.

Test automation is not easily achieved and not always the best choice, c.f. [38] chapter „Test Automation“. Due to the associated cost, it has to be wisely chosen.

Moreover, test automation has feedback effects on the plant models required for the simulation. Test automation typically follows known paths of stimuli. Therefore, the plant models need to represent valid behavior only in the spectrum of these stimuli. On the other hand, manual test execution requires much smaller investments in terms of test operation – but the test sequences are not known upfront. Therefore, the plant models must be universally applicable in the whole range of allowed user interaction during the test. This is a very high requirement and leads to extensive model verification cost.

Chapter 7 Evaluation of extra-functional properties

7.1 Design goals for evaluation of extra-functional properties

In this section, design goals for the evaluation of extra-functional properties are described. Examples for such properties are the behavior in case of faults or the timing of the simulation. To evaluate them, simulation based fault-injection will be used.

Simulation based fault-injection

As the validation framework is designed for distributed co-simulation, software and hardware implemented fault-injection will not be used. However, simulation based fault-injection can be integrated into the co-simulation framework and utilized during the simulations.

Interface and commands for fault-injection

To inject faults in the framework, an interface must be provided. Depending on the use-case, defined injection commands control the fault-injection.

Controllability of fault-injection (Determinism and reproducibility)

Controllable fault-injection has the same advantages like deterministic and reproducible co-simulation. The test cases can be repeated with the same system behavior.

Interface to monitor the system behavior

The goal of fault-injection is to analyze the system behavior in different fault-cases. To collect the required data, a monitoring interface in the framework is required.

Usage of results for analysis

Data collected by the monitoring interface shall be usable for analysis and provided in a defined format. For example, the mean times to and between failures (MTTF/MTBF) can be analyzed, respectively.

Support for different injection techniques

Nowadays, there are multiple injection techniques such as saboteurs, mutants or command-based techniques. The validation framework shall provide several of those techniques.

Catalog for different failure modes, failure types, etc.

A catalog shall be created listing all the failure modes, failure types, etc. supported by the validation framework. The user can decide which faults to inject based on this catalog.

Automated test-case generation

Due to the diversity of the different failure types and failure modes and the necessity to run multiple simulations, manual test case generation is time-consuming and costly. Hence, the generation process shall be automated.

Efficiency and performance

Since there are many simulation runs required to collect sufficient data for analysis, the fault-injection must be efficient and performant in the execution.

7.2 Introduction to fault-injection

Safety-critical environments often have ultrahigh-reliability requirements. Since the hardware reliability of VLSI (Very Large Scale Integration) components cannot reach ultrahigh reliability, fault-tolerance mechanisms become necessary (see [56], ch.11.4.1). Those mechanisms detect errors, locate their cause and recover into a non-faulty state. [80]

Although a major part of today's systems already contains fault-tolerance mechanisms, those mechanisms are not tested during quality assurance. Often, there is a lack of time or missing software tools and techniques such that only the regular functionality is in the scope of the final acceptance tests. As a result, the majority of causes for failing commissioning goes back to issues in the control software. [57]

Validating a dependable computer system should provide (i) a measure of the system's ability to detect errors, locate them and recover, (ii) a measure of the tolerance mechanisms' effectiveness, (iii) feedback to the engineer at the development stage for improvements and (iv) confidence in the system at design-time. An effective approach to evaluate the system's dependability properties and its fault-tolerance mechanisms is fault-injection [80]. The following sections will present different types of fault-injection techniques and tools implementing them.

7.3 Timing and reliability evaluation by fault-injection

Regarding different types of faults, it is useful to analyze different kinds of fault-injection techniques [74]. Those techniques can be categorized in three types according to the injection targets: (i) hardware implemented fault-injection (HWIFI), (ii) software implemented fault-injection (SWIFI) and (iii) simulation based fault-injection (SBFI). In the following, a brief overview about the various techniques is given.

7.3.1 Hardware implemented fault-injection

HWIFI uses additional hardware to inject faults. There are two injection types depending on the faults and their location, the injection with and without contact. While the first type has direct physical contact to the target system, the second one produces physical phenomena like radiation.

Both types induce voltage or current in the system to cause faults in it. Altering electric currents with contact is applicable using either active probes attached to pins or sockets. Indeed, active probes may destroy the device due to an inappropriate amount of current. Sockets are placed between the target hardware and its circuit board. They can invert pin signals or apply boolean operators like AND or OR on the signals.

Emulating physical phenomena does not need direct contact if radiation or electromagnetic interference is used. However, these techniques are not precisely controllable which is why it is difficult to exactly trigger the injection's time and its location. [44]

7.3.2 Software implemented fault-injection

Injecting faults in the target system enables the analysis of the system behavior in a large number of cases. The injection can cover all types of faults and failures in the system occurring at run-time systematically. Using SWIFI is scalable and its results are statistically verifiable. [91]

In contrast to HWIFI, SWIFI does not require expensive hardware and is applicable for target applications and operating systems (OS). In case of a target application, a fault-injection layer is added between the application and the OS. Adding a layer between the OS and the hardware is difficult to achieve. Hence, the fault injector must be embedded in the OS if it is the injection target. [44]

SWIFI is a flexible injection type but has some disadvantages. First, it cannot inject faults to locations inaccessible by software. Furthermore, the workload running on the target system may be disturbed and the structure of the original software might be changed. That is why software injection must be designed carefully. At last, the approach has a limited time resolution which may cause fidelity problems. While long latency faults (in memory, e.g) are not affected, faults with a short latency (in the CPU, e.g) might not be captured. One solution to this problem is a hybrid approach which combines software injection with hardware monitoring. [44]

Methods for SWIFI can be categorized in compile-time and run-time injection. The first technique changes the program instruction at design-time to generate an erroneous software image. When the system executes, the fault is activated. It is simple as it does not require additional software, but it does not allow injection at run-time. [44]

The injection at run-time requires a mechanism to trigger the injection. Such mechanisms can be a time-out, an exception/trap or a code insertion. To use the time-out, a software or a hardware timer is connected to the system's interrupt handler. As soon as the timer expires, an interrupt is invoked. This technique is easy to implement, but it is non-deterministic as the injection does not arise based on a specific event or system state. Compared with this approach, the exception/trap injects the fault whenever certain events or conditions occur. Similar to the time-out, control is passed to the interrupt handler. In the code insertion technique, instructions for fault-injection are added to the target program. Those allow the injection before a particular instruction. Furthermore, they can be executed in user-mode rather than the system mode which is required for a trap. [44]

7.3.3 Simulation based fault-injection

SBFI is useful to evaluate the effectiveness of fault-tolerant mechanisms and the system's dependability since it provides feedback to the developer. Indeed, the accurate inputs the mechanism requires are difficult to supply. One reason is that past measurements often cannot be used anymore when technology changes. However, precise results are provided by testing a prototype without any assumptions about the system design. [44]

The technique has been widely used, as it is simple, versatile and controllable [74]. Additionally, the fault-behavior as well as the fault-propagation is highly observable [92]. SBFI can be divided into three types. Those are the simulator command technique, the simulation model modification technique and the simulator modification injection. [74]

Injecting faults in the simulator command technique is done by built-in commands of the simulator. The faults manipulate the values of signals or variables in the target simulation model. It is a simple mechanism as does not require modifications in the simulation model. Indeed, it has limits in representing various fault-models. Additionally, performing the enormous number of fault-injection simulations to obtain acceptable results is inconvenient. [74]

Regarding simulation model modification, the saboteur and the mutant techniques are most widely adopted. Saboteurs are separate modules which inject various types of faults in the target and control the simulation [74]. They are capable of altering values as well as the system's timing characteristics. During normal execution, saboteurs remain idle while they are active in the fault-injection [92]. To inject faults, the saboteur contains an interface which is connected to the inputs and outputs of the target system. Hence, it requires modifications in the original target by adding a large number of control signals. This increases the complexity but there are more fault-models implementable than in the simulator command technique. Mutants are modified or corrupted modules representing the fault-injection. They are used to replace the original target and support various types of fault-models. Certainly, each mutant representing an error has to be prepared in advance which requires a large amount of storage area if many mutants need to be created. Typically, this is the case in any non-trivial fault-injection experiment.

Modifying the simulation kernel is used in the third technique. One advantage is that it must not change the original semantics of the target if an event-driven simulation engine is used. Hence, the experiments are simpler and more convenient than those in the simulation modification technique. Additionally, it requires less simulation resources because it can easily perform deterministic and random fault-injections. For faster reliability analysis by parallel execution, a distributed system can be used. [92]

7.3.4 Fault-injection tools

Since fault-injection is an efficient way to evaluate the system behavior in fault-cases, there are many tools available using the above-mentioned techniques or combinations of them. In this section, a portion of these tools is presented.

The Messaline tool [6] uses active probes and sockets to conduct pin-level fault-injection. Among others, the tool can inject stuck-at, open, bridging and complex logic faults and can control the length of fault-existence as well as its frequency. Additionally, collecting signals from the target system provides feedback to the user. The tool was used in experiments in a railway control system and in a distributed system.

FIST (Fault-injection System for Study of Transient Fault Effect) [37] creates transient faults inside a target system utilizing both contact and contactless techniques. It is capable of injecting faults directly inside a chip which is impossible using pin-level injection.

A different approach for contactless fault-injection is followed by MARS (Maintainable Real-Time System) [53] which is a distributed, fault-tolerant real-time architecture. In it, a circuit board is placed between two charged plates and antennas accentuate the electromagnetic field effects on individual chips.

One example for SWIFI is the Ferrari tool (Fault and Error Automatic Real-Time Injection) [52] which applies traps and trap handling. Traps are triggered by the program counter or a timer and the trap handling routines inject the faults as a consequence by e.g. manipulating registers or memory. Those faults can be of transient or permanent character.

Using Ftape (Fault Tolerance and Performance Evaluator) [100], engineers can likewise inject faults into registers and the memory. However, the tool emulates I/O-errors or bit-flips by related drivers in the operating system.

Doctor (Integrated Software Fault-injection Environment) [40] allows fault-injection for CPU, memory and the communication interface using three different ways. Those are time-outs, traps and code modification. Using the tool, the effect of intermediate message losses in distributed real-time systems can be analyzed.

Many modern processors have features for debugging and monitoring which are utilizable to inject more realistic faults. Those features are exploited in Xception [18] without any modification in the software. The fault injector acts like an exception handler and reacts to interrupts. Each interrupt is triggered by the processor based on a defined address. Hence, the experiments are reproducible.

Schlingloff and Vulinovic [91] introduce a model driven approach to evaluate the reliability of embedded controllers. Their tool allows the injection of faults on different modeling levels like the functional and the implementation model. Additionally, it supports various injection techniques like saboteurs or mutants.

VERIFY [92] is a software-based fault-injector which extends the VHDL language by fault-injection signals. In this way, the signals act as an interface for the simulator to enable the faults and each fault can be associated to a component. The tool supports dependability evaluation in all hardware design phases. Hence it is possible to analyze the rates of system crashes and recoveries on the one hand and the recovery time distribution which is related to the location or the type of the injected fault on the other one.

In the automotive and industrial automation domains, the approach of EffektiV [83] proves safety using virtual prototypes. Existing simulation environments are extended by fault-injection, monitoring and co-simulation mechanisms. Utilizing the tool in all design-stages supports decisions during the entire design-progress and safety-mechanisms can be evaluated. Additionally, the support of co-simulation possibilities enable the integration of further models in the experiments.

Na and Lee [74] propose the Verilog-based simulated fault-injection (VFI) technique. While mutants and saboteurs are inefficient and suffer from difficult experimental setups, VFI has various advantages. It is unnecessary to modify the design model and the simulation procedure is simple and efficient. Furthermore, this technique supports various types of faults. Building a VFI environment using the ICARUS Verilog simulator enables the evaluation of the approach's effectiveness. The simulation results prove the reliability of the target system as well as the effectiveness of the fault-tolerance mechanisms.

GOOFI (Generic Object-Oriented Fault-injection Tool) [4] is a generic tool supporting different target systems and fault-injection techniques. Since it is implemented in Java and relies on an SQL compatible database it is highly portable between different host platforms. To develop a user-friendly environment, it provides a graphical user interface and a generic architecture. The object-oriented approach further increases extensibility and maintainability of the tool. Next to pre-runtime SWIFI which injects errors before the simulation start, GOOFI supports Scan-Chain Implemented fault-injection. This technique exploits scan-chains available in VLSI circuits to inject faults at pins and in internal state elements. Moreover, the internal state can be observed.

Deterministic simulations are supported by FAUmachine presented in [88] which is a virtual machine for the simulation of standard PC hardware in cooperation with an environment. Since the tool uses just-in-time compilers, it provides good performance and can be used for reproducible tests in a reasonable time. Furthermore, FAUmachine allows testing for real-time constraints and provides the opportunity of multiple, automatically executed test-cases. To reach these features, the tool requires a real-time clock and uses its own cooperative light-weight thread-concept which ensures a predictable scheduling.

The framework presented in [16] is designed for in-depth analysis of transient faults and based on a debug-like mechanism to support analysis from an application point of view. It supports the designer to evaluate and to tune the system's dependability-related properties. The authors implement the framework in a fault-injection environment for SystemC and show that it is portable to other environments.

Rohani and Kerkhoff [87] propose a technique to reduce the consumption of CPU time in simulation-based fault-injection in complex System-on-Chip. It is compliant with commercial HDL simulators hence it does not require dedicated compilers. Additionally, the top-level modules do not have to be modified and a wide range of fault-models can be injected. The reduction of CPU time is achieved by utilizing simulator-commands along with partial code modification using saboteurs. During execution, the saboteurs are statically scheduled and completely controlled by simulator commands. Experimental results show a reduction of 27% to 67% in CPU time consumption compared to typical simulation-based fault-injection approaches.

The authors of [39] introduce two approaches to overcome crucial problems of software-based fault-injection techniques. While the first approach improves accuracy, the second one supplies detailed insight into the system dynamics in case of a fault. The insight enables to investigate the effect of faults and errors and fault-injection targets can be determined in time and location. Using this knowledge, the approach can significantly reduce the number of fault-injections since only the important cases are covered. Furthermore, both approaches can be joined together to combine their advantages.

Since fault-injection experiments must cover many cases, an automated test-case generation tool is desirable. Kormann and Vogel-Heuser [57] present such an approach to verify the

system's behavior in the presence of faults. They focus on a reduced set only covering meaningful, different test-cases. The complete set is generated automatically and is optimized in relation to its quantity. Only if predetermined fault detection and handling criteria are strictly met, a test case will be used. Additionally, test-cases traversing similar paths with different inputs are clustered which outperforms many other existing approaches. Hence, the approach scales depending on the desired degree of coverage.

7.4 Research gap for state-of-the-art

In this section, many different fault-injection tools are presented. Using those tools, properties of the system like the timing behavior or the behavior in case of faults can be observed. Furthermore, the simulation results enable to determine characteristics like the mean-time-to-failure or the mean-time-between-failure.

Since one design goal for the evaluation of extra-functional properties is the usage of simulation based fault-injection, many tools presented in the section are not applicable for this framework. They are either used for software or for hardware implemented fault-injection. However, the approaches presented in [72] and [74] both address efficiency and performance while [72] and [91] provide automated test-case generation. Additionally, all three tools are capable of injecting faults by using different injection techniques.

The presented fault-injection tools supporting simulation-based fault-injection have common properties related to the requirements defined in section 7.1. Most of the tools provide the required interfaces for both fault-injection and monitoring of the system's behavior. Furthermore, the injection is controllable in most cases to provide determinism and reproducibility and the monitoring results are usable for analysis. However, only VERIFY [92] and the approach presented in [87] address efficiency and performance while only [16] supports automated test-case generation. The injection by different techniques is realized in the different approaches presented in [83], [4] as well as [39]. A catalog with different failure modes or failure types is contained in none of the references cited.

On these grounds, the research gap for the evaluation of extra-functional properties is defined as follows. One fault-injection approach covering most of the requirements will be selected. Its capabilities need to be extended by the techniques related to the remaining requirements and afterwards the resulting mechanism is integrated into the evaluation and validation framework. Furthermore, the catalog with different failure modes and failure types has to be prepared.

Chapter 8 Summary and conclusion

One crucial development step in the railway domain is the integration of railway components and their testing. This process can be radically improved compared to today's practice by using a dedicated distributed simulation and validation framework. Therefore, one objective of the ongoing European research project Safe4RAIL lies in the concept design and a proof-of-concept implementation of such a framework. The goal is a network-centric simulation at system level providing a time-accurate simulation of in-train communication networks with co-simulated end-systems. Hence, simulators and physical systems shall be securely coupled via public communication networks.

In this deliverable, a SotA analysis of distributed simulation frameworks and other solutions of interest is presented. Such other solutions are Software- and Hardware-In-The-Loop simulation frameworks or tools to simulate wireless railway networks among others. The deliverable presents existing tools and defines the initial requirements for the design of the framework. Based on those aspects, the research gaps related to the topics can be defined.

Using a distributed SIL and HIL simulation framework leads to the handling of disturbances. In those cases, the communication can be delayed while the sensitivity for the simulation's quality depends on the disturbance itself. Furthermore, the simulation setup must be efficient and performant to provide fast SIL execution while also enabling real-time behavior for HIL.

One use case for the application of the framework is the testing of MCGs in the T2G communication. For this, a test environment will be developed in which the MCG provides the interface for the TCMS to communicate with the ground. The related peer will be the GCG. Both units will be integrated in the framework using HIL and SIL.

The combination of the HLA together with the FMI standard is a promising approach to interconnect simulations and physical devices in a distributed co-simulation framework. While FMI is already supported by various simulation tools, the HLA maintains the possibility to interconnect and synchronize the tools. Further solutions cover the requirements related to distributed co-simulation, timing, configuration and security of the framework.

Combining parts of the DO-330 / ED-215 standards with EN 50128 for the usage in the railway domain is important to validate the safety of the application. Hence, the standards will be probed and analyzed to report their applicability for the framework.

Since test automation is commonly used, hence there is no related research gap. However, test automation is not easily achieved and has feedback effects on the plant models in the simulation. The universal applicability of the plant models for user interaction during the tests is a very high requirement and leads to extensive model verification cost.

At last, evaluating extra-functional properties such as the timing behavior or the behavior in case of faults is another important aspect for the simulation and validation framework. To fulfill all requirements, a combination of the different tools presented in Chapter 7 is required. Finally, the developed mechanism needs to be integrated in the distributed co-simulation framework.

To conclude, there are already various approaches and standards available which can be utilized for the distributed simulation and validation framework. However, important challenges such as providing real-time execution in a distributed framework over the Internet, developing the T2G test environment or the safety validation still remain. In the following months, the initial requirements and design goals defined in this document have to be refined resulting in the final requirements for the project.

Chapter 9 List of Abbreviations

Table 9-1 List of Abbreviations

3G	Third generation of wireless mobile telecommunications technology (In Europe mostly UMTS)
4G	Fourth generation of wireless mobile telecommunications technology (In Europe represented by LTE)
5G	Fifth generation of wireless mobile telecommunications technology - proposed next telecommunications standards beyond the current 4G
A/D	Analog/Digital
ADN	Active Distribution Networks
AE	Authenticated Encryption
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AIK	Attestation Identity Key
AMS	Alpha Magnetic Spectrometer
API	Application Programming Interface
ASAM	Association for Standardisation of Automation and Measuring Systems
ATO	Automatic train operation
BER	Bit Error Rate
BSI	Bundesamt für Sicherheit in der Informationstechnik
CA	Certificate Authority
CACSD	Computer-Aided Control System Design
CBC	Cipher Block Chaining
CBTC	Communications-Based Train Control
CCTV	Closed Circuit Television

CMAC	Cipher-based Message Authentication Code
COS	Customer Oriented Service
COTS	Commercial Off-The-Shelf
CPS	Cyber-Physical System
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CS	Co-Simulation
CVC	Card Verifiable Certificate
D/A	Digital/Analog
DER	Distributed Energy Resources
DES	Data Encryption Standard
DHIL	Distributed Hardware-in-the-loop Simulation
DSA	Digital Signature Algorithm
DSIL	Distributed Software-in-the-loop Simulation
DSML	Domain-Specific Modeling Language
DUT	Device Under Test
ECC	Elliptic Curve
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECN	Ethernet Consist Network
ECRYPT	European Network of Excellence for Cryptology
ED	End Device
EdDSA	Edwards-curve Digital Signature Algorithm
EK	Endorsement Key
EPC	Evolved Packet Core
ERTMS	European Rail Traffic Management System
ETB	Ethernet Train Backbone

ETCS	European Train Control System
FER	Frame Error Rate
FMI	Functional Mockup Interface
FMU	Functional Mockup Units
FOM	Federation Object Model
FTP	File Transfer Protocol
GCG	Ground Communication Gateway as specified in IEC 61375-2-6
GME	Generic Model Environment
GSM	Global System for Mobile Communications
HIL	Hardware-In-The-Loop
HILS	HIL Simulation
HILSF	HIL Simulation Framework
HLA	High Level Architecture
HMAC	Hash-based Message Authentication Code
HSM	Hardware Security Module
HTTP	Hyper Text Transfer Protocol
HWIFI	Hardware Implemented Fault-Injection
I/O	Input/Output
IaaS	Infrastructure-as-a-Service (means enterprise IT infrastructure, e.g. virtual servers)
ICMP	Internet Control Message Protocol
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IGMP	Internet Group Management Protocol
IP	Internet Protocol
ISO	International Organization for Standardization

ISS	International Space Station
KPI	Key Performance Indicator
LLC	Logical Link Control
LTE	Long Term Evolution
MAC	Message Authentication Code
MAC	Media Access Control
MANET	Mobile Ad Hoc Networks
MCG	Mobile communication gateway as specified in IEC 61375-2-6
MD5	Message-Digest Algorithm 5
ME	Model Exchange
MIL	Model-In-The-Loop
MIMO	Multiple-input Multiple-output
MPI	Message Passing Interface
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OEM	Original Equipment Manufacturer – more specific in the scope of this document: supplier of an equipment for railway vehicle manufacturers
OMT	Object Model Template
OMTS	On-board Multimedia and Telematics System
ONVIF	Open Network Video Interface Forum
OOS	Operator Oriented Services
OS	Operating System
OTA	Over-the-air
PaaS	Platform-as-a-Service
PDCs	Phasor Data Concentrators
PHIL	Power HIL
PIS	Passenger Information Services

PK	Public Key
PKC	Public-Key Cryptography
PMUs	Phase Measurement Units
PoE	Power over Ethernet
PSS	Probabilistic Signature Scheme
PUF	Physical Unclonable Function
QoS	Quality of Service
RSA	Rivest-Shamir-Adleman
RT	Real-Time
RTDS	RT Digital Simulator
RTI	Runtime Infrastructure
RTSs	Real-Time Simulators
SBFI	Simulation Based Fault-Injection
SE	State Estimator
SHA	Secure Hash Algorithm
SK	Secret Key
SNMP	Simple Network Management Protocol
SotA	State-of-the-Art
SRAM	Static Read Access Memory
SRK	Storage Root Key
SSH	Secure Shell
SUT	System Under Test
SWIFI	Software Implemented Fault-Injection
T2G	Train-to-Ground
T2GTE	Train-to-Ground Test Environment
TCMS	Train Control and Monitoring System
TCN	Train Communication Network

TCP	Transmission Control Protocol
TD-LTE	Time Division Long Term Evolution
TLS	Transport Layer Security
TPM	Trusted Platform Module
TRDP	Train Real-Time Data Protocol
TTDP	Train Topology Discovery Protocol
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunication System
VoIP	Voice over Internet Protocol
WiFi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
WNCS	Wireless Networked Control Systems
XCBC	eXtended Ciphertext Block Chaining

Chapter 10 Bibliography

[1] New dependable rolling stock for a more sustainable, intelligent and comfortable rail transport in europe, d2.1 - specification of the wireless tcms. Technical report, Roll2Rail Project (Horizon2020).

[2] Eb assist adtf, December 2016.

[3] Marina Aguado, Oscar Onandi, Purificacion Agustin, Marivi Higuero, and Eduardo Taquet. Wimax on rails. *IEEE Vehicular Technology Magazine*, 3(3):47–56, 2008.

[4] Joakim Aidemark, Jonny Vinter, Peter Folkesson, and Johan Karlsson. Goofi: Generic object-oriented fault injection tool. In *Dependable Systems and Networks, 2001. DSN 2001. International Conference on*, pages 83–88. IEEE, 2001.

[5] Alexandre Amory, Fernando Moraes, Leandro Oliveira, Ney Calazans, and Fabiano Hessel. A heterogeneous and distributed co-simulation environment [hardware/software]. In *Integrated Circuits and Systems Design, 2002. Proceedings. 15th Symposium on*, pages 115–120. IEEE, 2002.

[6] Jean Arlat, Yves Crouzet, and J-C Laprie. Fault injection for dependability validation of fault-tolerant computing systems. In *Fault-Tolerant Computing, 1989. FTCS-19. Digest of Papers., Nineteenth International Symposium on*, pages 348–355. IEEE, 1989.

[7] M. Arnold, C. Clauß, and T. Schierz. Numerical aspects of fmi for model exchange and co-simulation v2.0. *Proc. of The 2nd Joint International Conference on Multibody System Dynamics, Stuttgart, Germany, May 29 - June 1, 2012*, 2012.

[8] Modelica Association. Fmi homepage. Website. Online available at https://fmi-standard.org/; accessed on 30.11.2016.

[9] Modelica Association. Functional mock-up interface, October 2016.

[10] Muhammad Usman Awais, Peter Palensky, Atiyah Elsheikh, Edmund Widl, and Stifter Matthias. 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*, pages 315–320. IEEE, 2013.

[11] Muhammad Usman Awais, Peter Palensky, Wolfgang Mueller, Edmund Widl, and Atiyah Elsheikh. Distributed hybrid simulation using the hla and the functional mock-up interface. *Industrial Electronics Society, IECON*, pages 7564–7569, 2013.

[12] Silvio Baccari. Real-time hil in railway: Simulations for testing control software of electromechanical train components. In Francesco Flammini, editor, *Railway Safety, Reliability and Security*. IGI Global, 2012.

[13] D Bian, M Kuzlu, M Pipattanasomporn, S Rahman, and 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*, pages 1–5. IEEE, 2015.

[14] Torsten Blochwitz, Martin Otter, Johan Akesson, Martin Arnold, Christoph Clauss, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar 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, pages 173–184. Linköping University Electronic Press, 2012.

[15] Torsten Blochwitz, Martin Otter, Martin Arnold, Constanze Bausch, H Elmqvist, A Junghanns, J Mauß, 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 Univeristy; Dresden; Germany*, number 063, pages 105–114. Linköping University Electronic Press, 2011.

[16] Cristiana Bolchini and Antonio Miele. An application-level dependability analysis framework for embedded systems. In *2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pages 171–178. IEEE, 2011.

[17] David Broman, Christopher Brooks, Lev Greenberg, Edward A Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. Determinate composition of fmus for co-simulation. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, page 2. IEEE Press, 2013.

[18] Joao Carreira, Henrique Madeira, João Gabriel Silva, et al. Xception: Software fault injection and monitoring in processor functional units. *Dependable Computing and Fault Tolerant Systems*, 10:245–266, 1998.

[19] Jon Casasempere, Pedro Sanchez, Tomas Villameriel, and Javier Del Ser. Performance evaluation of h. 264/mpeg-4 scalable video coding over ieee 802.16 e networks. In *2009 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, pages 1–6. IEEE, 2009.

[20] CENELEC. Functional safety of electrical/electronic/programmable electronic safety-related systems, 2010.

[21] CENELEC. Railway applications - communication, signalling and processing systems - software for railway control and protection systems, 2011.

[22] Michael Charitos and Grigorios Kalivas. Wimax-wlan vehicle-to-infrastructure network architecture during fast handover process. In *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pages 431–436. IEEE, 2013.

[23] Selim Ciraci, Jeff Daily, Jason Fuller, Andrew Fisher, Laurentiu Marinovici, and Khushbu Agarwal. Fncs: A framework for power system and communication networks co-simulation. In *Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative*, page 36. Society for Computer Simulation International, 2014.

[24] ACOSAR Consoritum. Literature review in the fields of standards , projects, industry and science. d1.1, 2016.

[25] Fabio Cremona, Marten Lohstroh, Stavros Tripakis, Christopher Brooks, and Edward A Lee. Fide: an fmi integrated development environment. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 1759–1766. ACM, 2016.

[26] Judith S Dahmann, Richard M Fujimoto, and Richard M Weatherly. The dod high level architecture: an update. In *Simulation Conference Proceedings, 1998. Winter*, volume 1, pages 797–804. IEEE, 1998.

[27] Atiyah Elsheikh, Muhammed Usman Awais, Edmund Widl, and Peter Palensky. Modelica-enabled rapid prototyping of cyber-physical energy systems via the functional mockup interface. In *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2013 Workshop on*, pages 1–6. IEEE, 2013.

[28] Tulga Ersal, Mark Brudnak, Jeffrey L Stein, and Hosam K Fathy. Statistical transparency analysis in internet-distributed hardware-in-the-loop simulation. *IEEE/ASME transactions on mechatronics*, 17(2):228–238, 2012.

[29] EUROCAE. Software considerations in airborne systems and equipment certification, 2012.

[30] EUROCAE. Software tool qualification considerations, 2012.

[31] Imade Fahd Eddine Fatani, Yann Cocheril, Crépin Nsiala, Baptiste Vrigneau, Marion Berbineau, and François-Xavier Coudoux. Robust train-to-wayside video communications in tunnels using h. 264 error-resilient video encoding combined with multiple antenna systems. *Transportation Research Part C: Emerging Technologies*, 25:168–180, 2012.

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

[33] ASAM Association for Standardisation of Automation ans Measuring Systems. Asam xil, September 2014.

[34] Alfredo Garro and Alberto 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*, pages 9–16. Society for Computer Simulation International, 2015.

[35] QA Systems GmbH. Cantata - accelerate safety standards compliance for c & c++ with cantata automated unit & integration test, 2015.

[36] Ke Guan, Zhangdui Zhong, Bo Ai, Ruisi He, Binghao Chen, Yuanxuan Li, and Cesar Briso-Rodriguez. Complete propagation model in tunnels. *IEEE Antennas and Wireless Propagation Letters*, 12:741–744, 2013.

[37] Ulf Gunneflo, Johan Karlsson, and Jan Torin. Evaluation of error detection schemes using fault injection by heavy-ion radiation. In *Fault-Tolerant Computing, 1989. FTCS-19. Digest of Papers., Nineteenth International Symposium on*, pages 340–347. IEEE, 1989.

[38] Rajeev Gupta. *Test Automation and QTP: QTP 9.2, QTP 9.5, QTP 10.0 and Functional Test 11.0*. Pearson Education India, 2012.

[39] Jens Guthoff and Volkmar Sieh. Combining software-implemented and simulation-based fault injection into a single fault injection method. In *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on*, pages 196–206. IEEE, 1995.

[40] Seungjae Han, Kang G Shin, and Harold A Rosenberg. Doctor: An integrated software fault injection environment for distributed real-time systems. In *Computer Performance and Dependability Symposium, 1995. Proceedings., International*, pages 204–213. IEEE, 1995.

[41] C Harding, A Griffiths, and Hongnian Yu. An interface between matlab and opnet to allow simulation of wncs with manets. In *2007 IEEE International Conference on Networking, Sensing and Control*, pages 711–716. IEEE, 2007.

[42] Graham Hemingway, Himanshu Neema, Harmon Nine, Janos Sztipanovits, and Gabor Karsai. Rapid synthesis of high-level architecture-based heterogeneous simulation: a model-based integration approach. *Simulation*, page 0037549711401950, 2011.

[43] Kenneth Hopkinson, Xiaoru Wang, Renan Giovanini, James Thorp, Kenneth Birman, and Denis 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.

[44] Mei-Chen Hsueh, Timothy K Tsai, and Ravishankar K Iyer. Fault injection techniques and tools. *Computer*, 30(4):75–82, 1997.

[45] Francisco Huerta, Jorn K Gruber, Milan Prodanovic, and Pablo Matatagui. Power-hardware-in-the-loop test beds: evaluation tools for grid integration of distributed energy resources. *IEEE Industry Applications Magazine*, 22(2):18–26, 2016.

[46] Vector Software Inc. Whitepaper: Vectorcast: Softwareverifikation, prüfung und validierung von anwendungen im bahnverkehr., 2016.

[47] International Standardization Organization. ISO 26262: Road Vehicles - Functional Safety (part 1-10), 2011.

[48] (Federal Aviation Administration) Frédéric POTHON (ACG Solutions) Gérard LADIER (Aerospace Valley) Jean-Louis BOULANGER (CERTIFER) Jean-Paul BLANQUART (Astrium) Philippe QUERE (Renault) Bertrand RICQUE (Sagem Défense Sécurité) Jean GASSINO (Institut de Radioprotection et de Sûreté Nucléaire) Jean-Louis CAMUS (Esterel Technologies), Michael P DEWALT. Tool qualification in multiple domains: Status and perspectives, erts2 2014 contributive paper. http://acg-solutions.fr/acg/wp-content/-uploads/2014/09/ERTS2014_MultiStandardToolQualification_1.0.pdf licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License., 2014.

[49] Hailin Jiang, Victor CM Leung, Chunhai Gao, and Tao Tang. Mimo-assisted handoff scheme for communication-based train control systems. *IEEE Transactions on Vehicular Technology*, 64(4):1578–1590, 2015.

[50] Dr. Lawrence Augustin Jürgen Döring. Introduction into asam mcd, September 1999.

[51] Ali Kalakech, Marion Berbineau, Iyad Dayoub, and Eric Pierre Simon. Time-domain lmmse channel estimator based on sliding window for ofdm systems in high-mobility situations. *IEEE Transactions on Vehicular Technology*, 64(12):5728–5740, 2015.

[52] Ghani A Kanawati, Nasser A Kanawati, and Jacob A Abraham. Ferrari: A tool for the validation of system dependability properties. In *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, pages 336–344. IEEE, 1992.

[53] Johan Karlsson, Peter Folkesson, Jean Arlat, Yves Crouzet, Günther Leber, and Johannes Reisinger. Application of three physical fault injection techniques to the experimental assessment of the mars architecture. *Dependable Computing and Fault Tolerant Systems*, 10:267–288, 1998.

[54] Brian M Kelley, Philip Top, Steven G Smith, Carol S Woodward, and Liang 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*, pages 1–6. IEEE, 2015.

[55] Arwa Khayat, Mohamed Kassab, Marion Berbineau, Mohamed Amine Abid, and Abdelfettah Belghith. Lte based communication system for urban guided-transport: A qos performance study. In *International Workshop on Communication Technologies for Vehicles*, pages 197–210. Springer, 2013.

[56] Hermann Kopetz. *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Real-Time Systems Series. Springer Science & Business Media, Berlin Heidelberg, 2. aufl. edition, 2011.

[57] Benjamin Kormann and Birgit Vogel-Heuser. Automated test case generation approach for plc control software exception handling using fault injection. In *IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society*, pages 365–372. IEEE, 2011.

[58] Renuka M Kulkarni, Rohita P Patil, and Chidambar Rao Bhukya. Hardware–in–loop test bench based failure mode effects test automation. *technology*, 4(6), 2016.

[59] Jean-Marc Kwadjane, Baptiste Vrigneau, Charlotte Langlais, Yann Cocheril, and Marion Berbineau. Performance evaluation of max-dmin precoding in impulsive noise for train-to-wayside communications in subway tunnels. *EURASIP journal on wireless communications and networking*, 2014(1):1–12, 2014.

[60] Wook Hyun Kwon and Seong-Gyu Choi. Real-time distributed software-in-the-loop simulation for distributed control systems. In *Computer Aided Control System Design, 1999. Proceedings of the 1999 IEEE International Symposium on*, pages 115–119. IEEE, 1999.

[61] Loi Lei Lai, Chong Shum, Linyu Wang, WH Lau, Norman Tse, Henry Chung, KF Tsang, and Fangyan Xu. Design a co-simulation platform for power system and communication network. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3036–3041. IEEE, 2014.

[62] David M Lane, Gavin J Falconer, Geoph Randall, and Ian Edwards. Interoperability and synchronisation of distributed hardware-in-the-loop simulation for underwater robot development: issues and experiments. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 1, pages 909–914. IEEE, 2001.

[63] Edward A. Lee and Sanjit A. Seshia. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. Lee and Seshia, 1 edition, 2010.

[64] W Li, A Monti, M Luo, and Roger A Dougal. Vpnet: A co-simulation framework for analyzing communication channel effects on power systems. In *2011 IEEE Electric Ship Technologies Symposium*, pages 143–149. IEEE, 2011.

[65] Hua Lin, Santhosh S Veda, Sandeep S Shukla, Lamine Mili, and James 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.

[66] Y Liu, M Steurer, and P Ribeiro. A novel approach to power quality assessment: real time hardware-in-the-loop test bed. *IEEE transactions on power delivery*, 20(2):1200–1201, 2005.

[67] Igor Lopez, Marina Aguado, and Christian Pinedo. A step up in european rail traffic management systems: A seamless fail recovery scheme. *IEEE Vehicular Technology Magazine*, 11(2):52–59, 2016.

[68] Bin Lu, Xin Wu, Hernan Figueroa, and Antonello Monti. A low-cost real-time hardware-in-the-loop testing approach of power electronics controls. *IEEE Transactions on Industrial Electronics*, 54(2):919–931, 2007.

[69] M Luglio, C Roseti, G Savone, and F Zampognaro. Tcp performance on a railway satellite channel. In *Satellite and Space Communications, 2009. IWSSC 2009. International Workshop on*, pages 434–438. IEEE, 2009.

[70] É. Masson and M. Berbineau. *Wireless Communications for Railway Applications*. 2017.

[71] Juan-Carlos Maureira, Paula Uribe, Olivier Dalle, Takeshi Asahi, and Jorge Amaya. Component based approach using omnet++ for train communication modeling. In *Intelligent Transport Systems Telecommunications,(ITST), 2009 9th International Conference on*, pages 441–446. IEEE, 2009.

[72] Silvio Misera, Heinrich Theodor Vierhaus, and Andre Sieber. Fault injection techniques and their accelerated simulation in systemc. In *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, pages 587–595. IEEE, 2007.

[73] Wolfgang Muller and Edmund Widl. Using fmi components in discrete event systems. In *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2015 Workshop on*, pages 1–6. IEEE, 2015.

[74] Jongwhoa Na and Dongwoo Lee. Simulated fault injection using simulator modification technique. *ETRI Journal*, 33(1):50–59, 2011.

[75] Himanshu Neema, Jesse Gohl, Zsolt Lattmann, Janos Sztipanovits, Gabor Karsai, Sandeep Neema, Ted Bapty, John Batteh, Hubertus Tummescheit, and Chandraseka 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, pages 235–245. Linköping University Electronic Press, 2014.

[76] Khanh TP Nguyen, Julie Beugin, Marion Berbineau, and Mohamed Kassab. A new analytical approach to evaluate the critical-event probability due to wireless communication errors in train control systems. *IEEE Transactions on Intelligent Transportation Systems*, 2016.

[77] Rafidah Md Noor and Christopher Edwards. Qos-enabled improvements for the network mobility protocol. In *Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd*, pages 1–5. IEEE, 2010.

[78] Zaher Owda, Mohammed Abuteir, and Roman 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)*, pages 1–8. IEEE, 2015.

[79] Mario Paolone, Marco Pignati, Paolo Romano, Stela Sarri, Lorenzo Zanni, and Rachid Cherkaoui. A hardware-in-the-loop test platform for the real-time state estimation of active distribution networks using phasor measurement units. In *Proc. Cigré SC6 Colloquium*, 2013.

[80] P Prinetto, Alfredo Benso, Fulvio Corno, M Rebaudengo, M Sonza Reorda, A Amendola, L Impagliazzo, and P Marmo. Fault behavior observation of a microprocessor system through a vhdl simulation-based fault injection experiment. In *Proceedings of the conference on European design automation*, pages 536–541. IEEE Computer Society Press, 1996.

[81] Luca Pugi. Hil testing of on-board subsystems: Some case studies and applications. In Francesco Flammini, editor, *Railway Safety, Reliability and Security*. IGI Global, 2012.

[82] Dr. Rainer Rasche. Asam xil 2.0: Standard für ein durchgängiges testen im gesamten produktentstehungsprozess. In *7. Vector Congress*, 2014.

[83] S Reiter, M Becker, O Bringmann, A Burger, M Chaari, R Drechsler, W Ecker, T Kruse, C Kuznik, J Laufenberg, et al. Fehlereffektsimulation mittels virtueller prototypen.

[84] Patrick F Riley and George F Riley. Spades. ”a distributed agent simulation environment with software-in-the-loop execution”. In *Proceedings of the 2003 Winter Simulation Conference S. Chick, PJ Sánchez, D. Ferrin, and DJ Morrice, eds*.

[85] Patrick J Roache and Patrick M Knupp. Completed richardson extrapolation. *Communications in Numerical Methods in Engineering*, 9(5):365–374, 1993.

[86] José Rodrguez-Piñeiro, José A Garcá-Naya, Angel Carro-Lagoa, and Luis Castedo. A testbed for evaluating lte in high-speed trains. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 175–182. IEEE, 2013.

[87] Alireza Rohani and Hans G Kerkhoff. A technique for accelerating injection of transient faults in complex socs. In *Digital System Design (DSD), 2011 14th Euromicro Conference on*, pages 213–220. IEEE, 2011.

[88] Matthias Sand, Stefan Potyra, and Volkmar Sieh. Deterministic high-speed simulation of complex systems including fault-injection. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 211–216. IEEE, 2009.

[89] Ina Schieferdecker. Test automation with ttcn-3 - state of the art and a future perspective. In José Carlos Alexandre Petrenko, Adenilso Simao; Maldonado, editor, *Testing Software and Systems*, pages 1–13. Lecture Notes in Computer Science, 2013.

[90] Tom Schierz, Martin Arnold, and Christoph Clauß. 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*, number 076, pages 205–214. Linköping University Electronic Press, 2012.

[91] Bernd-Holger Schlingloff and Sasa Vulinovic. Zuverlässigkeitsprüfung eingebetteter steuergeräte mit modellgetriebener fehlerinjektion. 2005.

[92] Volkmar Sieh, Oliver Tschache, and Frank Balbach. Verify: Evaluation of reliability using vhdl-models with embedded fault descriptions. In *Fault-Tolerant Computing, 1997. FTCS-27. Digest of Papers., Twenty-Seventh Annual International Symposium on*, pages 32–36. IEEE, 1997.

[93] Aleksander Sniady, Jose Soler, Mohamed Kassab, and Marion Berbineau. Ensuring long-term data integrity: Etcs data integrity requirements can be fulfilled even under unfavorable conditions with the proper lte mechanisms. *IEEE Vehicular Technology Magazine*, 11(2):60–70, 2016.

[94] Aleksander Sniady, Morten Sonderskov, and José Soler. Volte performance in railway scenarios: Investigating volte as a viable replacement for gsm-r. *IEEE Vehicular Technology Magazine*, 10(3):60–70, 2015.

[95] Patrick Sondi, Marion Berbineau, Mohamed Kassab, and Georges Mariano. Generating test scenarios based on real-world traces for ertms telecommunication subsystem evaluation. In *International Workshop on Communication Technologies for Vehicles*, pages 223–231. Springer, 2013.

[96] Steffen Straßburger. On the hla-based coupling of simulation tools. In *Proceedings of the 1999 European Simulation Multiconference*, volume 1, pages 45–51, 1999.

[97] Wenhao Sun, Xudong Cai, and Qiao Meng. Testing flight software on the ground: Introducing the hardware-in-the-loop simulation method to the alpha magnetic spectrometer on the international space station. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 815:83–90, 2016.

[98] Evoke Technologies. How to design an effective test automation framework, October 2014.

[99] Stavros Tripakis. Bridging the semantic gap between heterogeneous modeling formalisms and fmi. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on*, pages 60–69. IEEE, 2015.

[100] Timothy K Tsai, Ravishankar K Iyer, and Doug Jewitt. An approach towards benchmarking of fault-tolerant commercial systems. In *Fault Tolerant Computing, 1996., Proceedings of Annual Symposium on*, pages 314–323. IEEE, 1996.

[101] FEDERAL AVIATION ADMINISTRATION U.S. DEPARTMENT OF TRANSPORTATION. Software approval guidelines. https://www.faa.gov/-documentLibrary/media/Order/Order_8110.49_Chg_1.pdf, 2011.

[102] Cheng-Xiang Wang, Ammar Ghazal, Bo Ai, Yu Liu, and Pingzhi Fan. Channel measurements and models for high-speed train communication systems: a survey. *IEEE Communications Surveys & Tutorials*, 18(2):974–987, 2015.

[103] Xiaoxuan Wang, Hailin Jiang, Tao Tang, and Hongli Zhao. The qos analysis of train-ground communication system based on td-lte in urban rail transit. In *Intelligent Rail Transportation (ICIRT), 2016 IEEE International Conference on*, pages 49–54. IEEE, 2016.

[104] Tao Wen, Xinnan Lyu, David Kirkwood, Lei Chen, Costas Constantinou, and Clive Roberts. Co-simulation testing of data communication system supporting cbtc. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 2665–2670. IEEE, 2015.

[105] Alan Facchinetti and Stefano Bruni. *Hardware-in-the-loop hybrid simulation of pantograph–catenary interaction*. In *Journal of Sound and Vibration*, Volume 331, Issue 12, 2012, pages 2783-2797, 2012.

[106] Peter Terwiesch, Thomas Keller and Erich Scheiben. *Rail vehicle control system integration testing using digital hardware-in-the-loop simulation*. In *IEEE Transactions on Control Systems Technology*, vol. 7, no. 3, pages 352-362, May 1999.

[107] J. N. Verhille, A. Bouscayrol, P. J. Barre and J. P. Hautier. Hardware-in-the-loop simulation of the traction system of an automatic subway. In *2007 European Conference on Power Electronics and Applications*, pages 1-9, Aalborg, 2007.

[108] J. N. Verhille, A. Bouscayrol, P. J. Barre and J. P. Hautier. Validation of anti-slip control for traction system using Hardware-In-the-Loop simulation. In *IEEE Vehicle Power and Propulsion Conference*, Arlington, TX, pages 440-447, 2007.

[109] Silvio Baccari, Giulio Cammeo, Christian Dufour, Luigi Iannelli, Vincenzo Mungiguerra, Mario Porzio, Gabriella Reale, and Francesco Vasca. *Real-Time Hardware-in-the-Loop in Railway:Simulations for Testing Control Software of Electromechanical Train Components*. In Railway Safety, Reliability, and Security: Technologies and Systems Engineering, pages 221-248, May 2012.