DISSERTATION

Interface Design for Hardware-in-the-Loop Simulation of Real-Time Systems

ausgeführt zum Zwecke der Erlangung des
akademischen Grades eines

Doktors der technischen Wissenschaften

unter der Leitung von

O.Univ.-Prof. Dr. Hermann Kopetz
und
Dr. Wilfried Elmenreich
als verantwortlich mitwirkendem Universitätsassistenten
Institut für Technische Informatik 182

eingereicht an der

Technischen Universität Wien
Fakultät für Informatik

von

Martin Schlager
Matr. - Nr. 9725343
2641 Schottwien 103

Wien, im September 2007 .....

Interface Design for Hardware-in-the-Loop Simulation of Real-Time Systems

Abstract

Hardware-in-the-Loop (HiL) simulation is a testing technique in which the environment of an (embedded) *System-Under-Test (SUT)* is simulated by an assigned HiL simulator. Thereby, the SUT interacts with the HiL simulator via the SUT's interface with its environment. The interaction between the SUT and the HiL simulator takes place in real time and is constrained by the temporal properties of the SUT. In the case where the SUT is a distributed system, consisting of several nearly-independent computers interacting with their environment in a collaborative way, the set-up of an HiL simulation is a non-trivial task requiring well-designed linking interfaces of the HiL simulator to enable predictable test runs.

This thesis proposes an approach towards the temporal decoupling of environmental simulation of the HiL simulator and the SUT, by using a time-triggered *connection system* acting as a *temporal firewall*. Using such an approach, it can be guaranteed that information flow between the HiL simulator and the SUT (and vice versa) is bound to *a priori* known latency and jitter. Furthermore, timing violations of an HiL simulation can be deterministically diagnosed and actions, avoiding detrimental consequences of such timing violations, can be initiated.

In contrast to traditional solutions to interfacing between the SUT and the HiL simulator, the presented approach allows *temporal laxity* of the HiL simulator, i. e., the execution of the simulation model (e. g., a MATLAB/Simulink model) is individually performed by assigned components of a distributed HiL simulator while different components of the same HiL simulator are responsible for timely interfacing with the SUT.

The actual physical coupling of the SUT and the HiL simulator is established via an arbitrary transducer interface. This interface can be implemented using a physical transducer, a (standardized) digital transducer interface, or a so-called *Smart Virtual Transducer (SVT)* that mimics the behavior of a physical transducer. The thesis provides an outline of a generic HiL simulation framework, based on SVTs. The proposed framework is exemplarily applied to the verification of integrated systems.

Schnittstellendesign für die Hardware-in-the-Loop Simulation von Echtzeitsystemen

Kurzfassung

Hardware-in-the-Loop (HiL) Simulation ist ein Testverfahren, bei der die Prozessumgebung eines zu testenden Systems (System-Under-Test (SUT)) durch einen HiL Simulator ersetzt wird. Dabei interagiert das SUT mit dem HiL Simulator über dieselbe Schnittstelle des SUTs, über welche das SUT mit der Umgebung interagieren würde. Diese Interaktion zwischen dem SUT und dem HiL Simulator findet in Echtzeit statt und hängt von den temporalen Eigenschaften des SUTs ab. Der Aufbau einer HiL Simulation erfordert daher einen sorgfältigen Entwurf der Schnittstellen, um vorhersagbare Testabläufe zu ermöglichen. Dies ist insbesondere dann der Fall, wenn es sich beim SUT um ein verteiltes System handelt, das sich aus mehreren voneinander (nahezu) unabhängigen, mit ihrer Umgebung interagierenden Computern zusammensetzt.

Diese Arbeit stellt einen Ansatz zur zeitlichen Entkopplung von Umgebungssimulation und SUT mittels zeitgesteuertem Adaptersystem (*Connection System*) und kontrollfehlerfreien Schnittstellen (*Temporal Firewall*) vor. Dadurch kann garantiert werden, dass der Informationsfluss von der Umgebungssimulation des HiL Simulators zum SUT (und vice versa) durch vorab bekannte Wartezeiten (*Latencies*) und Schwankungen (*Jitter*) bestimmt ist. Weiters können Zeitüberschreitungen (*Timing Violations*) einer HiL Simulation diagnostiziert, und negative Konsequenzen solcher Zeitüberschreitungen vermieden werden.

Im Vergleich zu traditionellen Lösungen vereinfacht der vorgestellte Ansatz die zeitlich korrekte Ausführung eines Simulationsmodells. So kann beispielsweise die (Schwankungen unterworfene) Berechnung eines MATLAB/Simulink Modells auf einer Komponente eines verteilten HiL Simulators ausgeführt werden, während weitere Komponenten für die zeitgerechte Interaktion mit dem SUT verantwortlich sind.

Die tatsächliche physikalische Anbindung des SUTs an den HiL Simulator kann über verschiedene Messwandlerschnittstellen (*Transducer Interfaces*) aufgebaut werden. Diese Schnittstellen können in Form eines physikalisch existierenden Messwandlers, einer (standardisierten) digitalen Messwandlerschnittstelle, oder einer so genannten *Smart Virtual Transducer (SVT)* Komponente realisiert werden. Diese Arbeit stellt ein – auf der Verwendung von SVTs basierendes – HiL Simulationssystem, sowie die Verifikation von integrierten Systemen unter Verwendung dieses Simulationssystems vor.

Danksagung

Ich möchte mich bei meinem Betreuer und Institutsvorstand O.Univ.Prof. Dr. Hermann Kopetz bedanken, der mir ermöglichte, diese Arbeit im äußerst interessanten Arbeitsfeld des Instituts für technische Informatik der TU Wien zu verfassen und mir in ausgesprochen konstruktiver Arbeitsatmosphäre mit vielen wertvollen Anregungen zur Seite stand.

Ein großes Dankeschön möchte ich insbesondere auch meinem Kollegen, Wilfried Elmenreich, aussprechen, der durch wichtige Ideen, zahlreiche Diskussionen und kritische inhaltliche Auseinandersetzung mit dem Thema einen wesentlichen Einfluss auf die vorliegende Arbeit hatte. Außerdem möchte ich mich bei meinem zweiten Begutachter, Univ.Prof. Dr. Johann Blieberger, für wertvolle Anregungen und Tipps zur vorliegenden Arbeit bedanken.

Danken möchte ich auch zahlreichen Kollegen am Institut, die durch sehr gute Gespräche und freundschaftliche Unterstützung zum Resultat dieser Arbeit beigetragen haben. Namentlich bedanken möchte ich mich besonders bei Roman Obermaisser für die exzellente Zusammenarbeit im Bereich der Validierung integrierter Systeme. Weiters danke ich Gerhard Burger, Florian Skopik und Michael Wihsböck für die Implementierung der Case Studies, sowie Wilfried Elmenreich, Marlene Kritz, Ingomar Wenzel und Martina Sebastian für das gründliche Korrekturlesen der Arbeit. Bei Maria Ochsenreiter möchte ich mich für die großartige organisatorische Unterstützung im Laufe der Jahre sehr herzlich bedanken.

Mein Dank gilt weiters der Österreichischen Akademie der Wissenschaften, die diese Arbeit im Rahmen eines einjährigen Dissertationsstipendiums (DOC) großzügig unterstützte, sowie meinen Vorgesetzten und Kollegen in der Firma TTTech Computertechnik AG, namentlich Judith Sattlberger, Stefan Poledna, Tibor Gajdoš und Georg Stöger, die mir insbesondere im Zuge der Fertigstellung dieser Arbeit mit einem Höchstmaß an Flexibilität entgegengekommen sind.

Bei meinen Großeltern, meinen Eltern, Waltraud und Franz, und meinen Geschwistern, Thomas und Nicole, möchte ich mich an dieser Stelle ganz besonders bedanken. Sie haben mir durch ihre Liebe über Jahrzehnte ermöglicht, die Grundlage für diese Arbeit zu schaffen und standen mir stets unterstützend zur Seite. Weiters möchte ich Ingomar Wenzel, Jacek Ratzinger, Daniela Woditschka, Barbara Huber, Gerhard Krizovsky, Alexander Rudolf, Bernhard Rumpler und Natascha Vecsera meinen Dank für eine großartige Zeit, schöne gemeinsame Hobbys und langjährige freundschaftliche Verbundenheit aussprechen.

Diese Arbeit ist meinem Großvater, August Spanring, gewidmet.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Within the last decades, major advances in the semiconductor industry [EKRZ04, Fro04] have enabled the widespread application of embedded real-time systems. These systems have become an immanent part of our daily lives and can be found in many different domains such as the manufacturing, aerospace, automotive, railway, and consumer electronics markets. At present, embedded real-time systems contribute to a significant share of the innovations within these markets.

Hardware-in-the-Loop (HiL) simulation is a commonly used technique for validating embedded real-time systems prior to their actual deployment. An HiL simulation is an operational configuration of a System-Under-Test (SUT) and its respective environment. The SUT is (part of) a real-time system and can be a single embedded controller or a distributed system.

The application of HiL simulation as a validation technique of an embedded real-time system thrives on the fact that HiL simulation is closer to reality than any other simulation technique (e. g., Software-in-the-Loop (SiL)). HiL simulation makes it possible to test the system in rare but potentially hazardous situations without building costly prototypes and without endangering humans or the natural environment.

The construction of sophisticated HiL simulation systems is particularly relevant for embedded real-time systems that are used for safety-related tasks, e. g., fly-by-wire. Safety-related real-time systems must guarantee correct behavior in the presence of faults – even if the occurrence of these faults is very unlikely. The aerospace industry is an important innovation driver for safety-related real-time systems. These systems have been used in airborne equipment for many years. Recently, the automotive industry has also started employing safety-related systems.

1.1 Motivation and Objectives

The development of a safety-related real-time system involves verification and validation techniques ensuring sufficient trust in the reliability of the system. As part of

validation, dynamic testing aims to remove errors within a system prior to its final deployment.

However, it is often difficult to test a real-time computer system performing control tasks in its natural environment because the conditions of the environment circumvent an accurate observation of the real-time systems interfaces. Furthermore, it may be very costly or even impossible to establish certain test conditions in the physical process environment of the real-time computer system.

Real-time simulation of the environment of a real-time computer system through a computer program as discussed within this thesis allows the observation and systematic modification of many relevant parameters. HiL simulation includes the actual physical real-time computer system and a simulation of the environment (of this real-time computer system). Thereby, an HiL simulator executes a model in real time which represents the behavior of the environment (i. e., traces of significant state changes at the interface between the environment and the real-time computer system). The interaction between the real-time computer system and the environmental simulation is constrained by properties of the real-time system, as well as by physical properties of the employed hardware.

HiL simulation is an approach that has been introduced by the aerospace and defense industries in the 1950s [NBAR04]. At this time, the high cost of HiL technology hindered its widespread use. During the past decades, accessibility of powerful computing resources to virtually every engineer and decreasing prices of simulation hardware led to adoption of HiL simulation to further domains such as industrial control applications or automotive systems.

Our aim is to introduce an HiL simulation framework that offers an instrument for the construction of predictable, cost effective, component based HiL simulators that can be applied in different industrial domains. A high potential for cost reduction through improved simulation and test techniques exists particularly in the automotive industry, which is due to large production quantity and enormous cost implications of recalls. Furthermore, decreasing development cycles, i. e., shorter time-to-market, and stringent cost limitations within this industry require novel development approaches in order to stay competitive [VDA03].

As outlined in [SEW06], the benefits of HiL simulation for the development of real-time systems can be summarized as follows:

- Testing of early system prototypes in a simulated environment becomes possible.
- An artificial environmental situation can be constructed that is in line with a defined test scenario.
- Effective monitoring is possible because control values from the SUT are received by the environment simulator. These control values can be further processed or recorded.
- Once a simulation is set-up, it is possible to perform a large number of tests without significant cost implications.

- It is possible to develop a real-time system and to perform tests, even if the environment, i.e., the controlled object, is not accessible during development.
- It is possible to test the behavior of a real-time system in hazardous situations. In the real environment it could be very costly (e. g., crash test) or not even feasible/acceptable (e. g., emergency actions of an aircraft during flight) to guide the system into such situations.
- HiL simulation supports regression testing, i. e., different versions of a real-time system can be tested using the same test conditions in order to uncover regression bugs.

1.2 Related Work

This thesis builds on two main areas of research: real-time systems and simulation. Both areas have been well covered within the scientific literature. Thus, a huge number of both, theoretical as well as application oriented publications is available.

A comprehensive summary on concepts regarding the design and analysis of real-time systems can be found in [Kop97, Liu00, Lap04]. With regard to dependable real-time systems, a fundamental study on the concepts of dependability is presented in [Lap92, ALRL04]. A system architecture for the construction of large dependable real-time systems is given by the Time-Triggered Architecture (TTA) [KB03]. The design of interfaces for real-time systems is tackled in [Krü97, KFMN99, KS03] and extensively discussed in the EU funded project Dependable Systems of Systems (DSoS) [AFI+00, JKK+02].

General concepts of simulation are comprehensively summarized in [Law06]. In [BCNN01] emphasis is put on discrete system simulation. Even the subarea of (real-time) HiL simulation and its application is very well covered in both, academia and industry, e. g., in [NI03, NBAR04, WLD+05, BSS05].

However, up to now there is a lack of research particularly focusing on interface design for HiL simulation. Current approaches largely target at improving the signal quality during an HiL simulation. For example various interfacing schemes for HiL simulation of power electronics applications are presented in [MFL+05].

In order to increase signal accuracy during an HiL simulation, a dual time step simulation method is proposed in [LGDL06, GLH06]. Thereby, a Field Programmable Gate Array (FPGA) is used with very small time steps for handling a switching circuit while the remaining simulation is executed by a conventional processor at a much larger time step. In [Jim05] an approach is presented that uses an averaging method to improve accurate acquisition of signals whilst generating realistic and fast feedback signals.

The HiL simulation methods mentioned in the previous paragraphs as well as commercially available products such as the dSpace Simulator[1] or the Opal RT-

[1] http://www.dspace.com

LAB[2] aim to improve the accuracy of the coupling between the simulator and the SUT by providing fast devices (FPGAs).

Furthermore, many HiL simulation approaches focus on testing a single controller and only few solutions exist that explicitly target at HiL testing of a large distributed real-time system. In chapter 3, section 3.3 an analysis of commercially available HiL simulators is provided.

Simulation of the environment of a time-triggered real-time system is addressed through the implementation of a distributed discrete time environment simulator as outlined in [Sch95], and by research on cluster simulation [FRB97, Gal99]. In [Sch95], the need for interface tests is mentioned with respect to interfaces used for the coupling between the environment simulator and the SUT. Such tests require well specified interfaces.

1.3 Contribution

This thesis outlines the structure and the constituting elements of an HiL simulation framework placing particular emphasis on predictable interaction between the HiL simulator and the SUT. Thus, a distinction between components of an HiL simulator performing a simulation of the environment of an SUT, and components that are being used to couple the HiL simulator and the SUT, is drawn.

The thesis elaborates on different approaches for establishing the physical coupling between SUT and HiL. Thereby, the new concept of a *Smart Virtual Transducer (SVT)* and the prototype implementation of an SVT are shown. An SVT is a hardware/software component employed to replace a physical sensor or actuator of the SUT. An SVT supports predictable interactions between an HiL simulator and an SUT via an arbitrary transducer interface.

Furthermore, the thesis outlines a possible application of the HiL simulation framework to large integrated systems. Therefore, it is exemplarily demonstrated how an HiL simulation can be used as part of an X-in-the-Loop testing process that encompasses Model-in-the-Loop (MiL), SiL, and HiL. Moreover, a simulation framework for virtual integration of an integrated system as part of SiL tests is presented.

In the presented HiL simulation framework, the execution of the simulation model (e. g., a MATLAB/Simulink model) is individually performed by assigned components of a distributed HiL simulator while different components of the same HiL simulator are responsible for timely interfacing with the SUT. As an advantageous result of this separation, the execution of the simulation model is temporally decoupled from precise instants of interaction with the SUT as long as bound maximum execution times of single simulation steps are guaranteed.

The thesis presents a distributed HiL simulator that is (1) based on a time-triggered communication and process model and that (2) involves a strict separation of the execution of the simulation model and the interaction between this simulation

[2] http://www.opal-rt.com

model and the SUT. The benefits of this approach are given as follows:

- The cognitive complexity of a simulation model (i. e., the amount of cognitive resources that are needed to understand the model [RS07]) is reduced by a decreased number of system variables and their dependencies. This is achieved by the introduction of loosely coupled HiL simulator components that interact across well-specified interfaces.

- It is possible to implement the HiL simulation model with discrete simulation methods that are in accordance to the non-blocking *simple task (S task)* [Kop97] execution scheme.

- Based on the concept of a *temporal firewall* [KN97], *temporal laxity* of the HiL simulator is facilitated. Temporal laxity of the HiL simulator means that jitter of a single simulation step does not influence the predictable instants of interaction between the HiL simulator and the SUT as long as the execution of a single simulation step is constrained by an upper-bound Worst Case Execution Time (WCET).

1.4 Structure of the Thesis

The remainder of this thesis is structured as follows:

Chapter 2 comprises an investigation on the basic terms and concepts that constitute the fundamentals of this thesis. Chapter 2 is partitioned into four subsections focusing on the relevant entities of real-time systems, interfaces, simulation, and Smart Transducers (STs), respectively.

Chapter 3 outlines the structure and basic elements of an HiL simulation system. Furthermore, requirements of an HiL simulation system including a distributed real-time system are described and existing HiL solutions and their respective possible applications are discussed.

Chapter 4 covers the essential part of this thesis which is the interface design for an HiL simulation. Within this chapter, an HiL simulation framework is introduced that offers a stable intermediate interface between SUT and HiL simulator. The concept of an SVT is outlined in detail which is used as an interfacing scheme for arbitrary transducer interfaces.

Chapter 5 deals with the application of the proposed HiL simulation framework for X-in-the-Loop testing of an integrated system. In this chapter X-in-the-Loop testing of an integrated system is explained that can be performed based on a – in context of this thesis developed – virtual integration platform. The HiL simulation framework is employed as a final pre-deployment step and exemplified within this chapter.

Chapter 6 describes three different case study setups. The first setup includes the realization of a digital ST gateway. The other case studies encompass HiL simulators based on SVTs that support (a) the simulation of a control path and (b) the development of a driver assistance system prototype.

Chapter 7 concludes the thesis by providing a summary of the main results and a discussion on potential future research work.

Chapter 2

Basic Terms and Concepts

2.1 Real-Time Systems

Throughout the past decades, many concepts and terms related to real-time systems have been introduced. In the literature, several definitions can be found that are tailored to the specific needs of a particular research area of interest. In the following section, we focus on fundamental concepts and the related taxonomy that is used throughout this thesis.

According to IEEE standard glossaries [IEE89, IEE90] a *system* can be defined as *a collection of components organized to accomplish a specific function or set of functions*. A more detailed definition can be found in [Lap92, ALRL04], where a system is regarded to be an entity that interacts/interferes with other entities, i. e., other systems. The set of these "other entities" that a system interacts/interferes with, is called the *environment* of the system. The environment may include hardware-software entities, humans, and the physical world with its natural phenomena. The system produces on the one hand output information that influence properties of the environment and measures on the other hand input information that is produced by the environment.

The definition of the term system in the context of real-time systems additionally requires the incorporation of the notion of time. The conceptual model developed in the EU funded project DSoS defines system as *an entity that is capable of interacting with its environment and may be sensitive to the progression of time* [JKK+02]. By "sensitive to the progression of time" is meant that the reaction of the system may depend on the progression of time, i. e., a given input pattern may result, due to the progression of time, in different output patterns.

Furthermore, in [JKK+02], the decomposition of a system into *interacting components*, that may themselves consist of interacting components, is being highlighted. The recursive process of decomposing a system into interacting components is stopped as soon as any further internal structure of an *atomic component* [ALRL04] is of no relevance or cannot be discerned for the current analysis.

With respect to the above given definitions, a *real-time system* is a system that

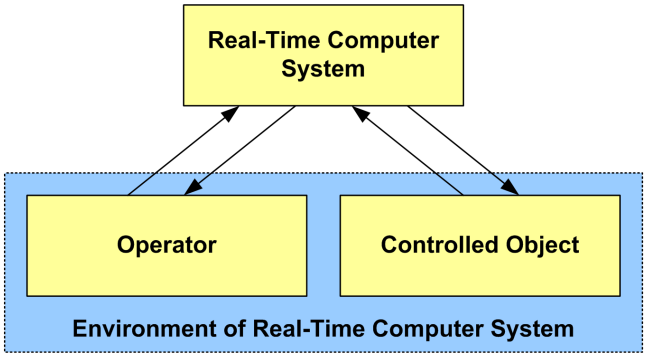

Figure 2.1: Real-time system

performs an action, e. g., the execution of a software task, the actuation of a brake in a car, or the visualization of data on a display unit, within an *a priori* defined time interval, i. e., before the expiration of a *deadline*. In accordance to the description given in [Kop97], the basic building blocks of a real-time system are: the *real-time computer system*, the *controlled object*, and the *operator* (refer to figure 2.1). The controlled object and the operator are regarded as the environment of the real-time computer system.

The *service* of a real-time computer system is a *type of operation that has a published specification of interface and behavior* [AUT06a], involving a contract between the real-time system's capability and its environment. In accordance to the DSoS conceptual model [JKK+02], *behavior* is defined as *traces of activity at system interfaces* which are *sequences of (perhaps timestamped) send and receive operations of the system.*

The service of a real-time computer system is provided to its environment via *transducers*, i. e., sensors and actuators that transform a physical stimulus into a signal and vice versa. The interaction of the real-time computer system with its environment, i. e., with the controlled object and with the operator, is bound to time intervals that are predetermined by the environment. Thus, the correct system behavior of a real-time computer system not only depends on the logical results of the computation (space/value domain), but also on the physical instants at which these results are produced (time domain) [Kop97].

An *application* is *a software (or program) that is specified to the solution of a problem of an end user requiring information processing for its solution* [AUT06a]. An application consists of a set of *functions*, i. e., *tasks, actions or activities that must be accomplished to achieve a desired outcome* [AUT06a]. We call an application that is executed as part of a real-time computer system a *real-time application.*

2.1.1 Classification of Real-Time Systems

Real-time systems can be classified in several ways. Kopetz [Kop97] draws a distinction between properties outside the real-time computer system (e. g., hard real-time versus soft real-time) and properties inside the real-time computer system

(e. g., guaranteed-response versus best-effort). In the following paragraphs we investigate a subset of these classifications.

Hard Real-Time Systems versus Soft Real-Time Systems

In the case that timing constraints given by the environment are not met, the performance of a real-time system may be degraded, or may even fail completely. Thus, we have to distinguish between:

- *soft real-time systems* where the usefulness of the results of the real-time system is degraded but not void if one or more (soft) deadlines are missed,
- *firm real-time systems* where the usefulness of the results of the real-time system is null as soon as a (firm) deadline is missed, and
- *hard real-time systems* where a catastrophe (e. g., plane crash) could be the result of a missed (hard) deadline [Kop97, Lap04].

A real-time system that has at least one firm deadline is called a *firm real-time system*. If the real-time system has at least one hard deadline, it is called *hard real-time system* or *safety-critical real-time system*.

Guaranteed-Response versus Best-Effort

When designing a real-time system, assumptions about the required computational resources have to be made. The *load hypothesis* [Kop97] states that these assumptions must be fulfilled by the real-time system in order to guarantee system response even in a peak-load scenario. Furthermore, a *fault hypothesis* [Kop04] is required, that defines the type and number of faults which the real-time system must be able to tolerate.

Depending on compliance with the load- and the fault-hypotheses, a distinction between two kinds of real-time system designs can be made:

System design with guaranteed response behavior: In case it is possible to argue that a real-time system design fully complies with the load- and the fault-hypotheses without any probabilistic arguments, the real-time system has a *guaranteed response behavior*. Careful planning and extensive analysis is required for such a system.

Best-effort system design: In the case where compliance with the load- and the fault-hypotheses is argued in terms of probabilistic arguments (e. g., , extensive tests) the real-time system has a *best-effort behavior*. In rare-event scenarios, the response of the real-time system may not comply to the load- and to the fault-hypotheses.

The differentiation between guaranteed response and best-effort relates to properties of the real-time computer system. The differentiation between hard- and soft real-time systems relates to requirements of the process environment of the real-time computer system. In order to validate a hard real-time system, a real-time computer system with a guaranteed response is required. At present, most soft real-time systems are designed in line with a best-effort system design.

Event-Triggered versus Time-Triggered Systems

All actions of a real-time computer system, e. g., task activation, or start of message transmission must be *triggered* in order to happen. A *trigger* is an event that causes the start of an action [Kop97]. With respect to the triggering mechanism, event-triggered and time-triggered real-time computer systems can be distinguished from each other.

Event-triggered: In an event-triggered real-time computer system, all system actions are triggered by the occurrences of events within the environment or within the real-time computer system [Kop93].

Time-triggered: In a time-triggered real-time computer system, all system actions are triggered by the progression of physical time [Kop93].

Real-time systems that follow the event-triggered paradigm require less effort during the planning and design phase, and usually offer more flexibility towards system alterations. In contrast, time-triggered systems provide predictable temporal behavior and composable system design, consequently being superior within the domain of safety-critical real-time systems.

2.1.2 Model of Time

The progression of time plays a key role in the operation of real-time systems. According to the Newtonian model that is adopted by most real-time applications, *real-time* is an infinite set of instants on a directed time line that goes from past to future [KS03]. An *instant* is a cut of the time line. A section of the time line between two instants is called *duration* or *interval*. An *event* is an occurrence that happens at an instance, e. g., sending a message, operating on a transducer, or activating a task.

In a *dense time base* [Kop92, Kop97], between any two instants there is at least a further instant. Due to the imprecision of physical clocks, slight deviations of clocks of different observers are unavoidable. Hence, it is impossible to guarantee that timestamps of two observers that watch the same event will equal. In a distributed real-time system correct ordering of two events can consequently not be guaranteed in case these two events occur within an interval that is smaller than the precision of the physical clocks.

Given a *sparse time base* [Kop92, Kop97], the occurrence of events is restricted to active intervals that are separated by intervals of silence (refer to figure 2.2). Events

happening during a given active interval are considered to happen at the same time. Of course, only events that are within the sphere of control of the real-time computer system can be restricted to occur in active intervals. External events, i. e., events that are not within the sphere of control of the real-time computer system, are based on a dense time base [JKK+02].

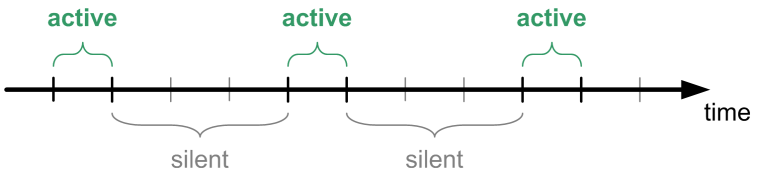

Figure 2.2: Sparse time base

2.1.3 Messages

The exchange of information within a real-time computer system is usually based on messages that are passed between interacting computing nodes of a distributed system (refer to section 2.1.5). Thereby, a *message* is *a data structure that is formed for the purpose of communication among computer systems* [JKK+02]. Even though the interaction between a (real-time) computer system and its environment may involve other approaches than message passing (e. g., direct analog signals), it is often possible to abstract the interaction between the real-time computer system and its environment to be solely based on the exchange of messages. Especially in the case where an ST system (refer to section 2.4) is employed, specific properties of the interaction between the real-time computer system and its environment are encapsulated within the ST system.

Based on application specific input/output assertions, i. e., claims regarding acceptance of a message for a certain application, messages can be classified into different categories (refer to table 2.1).

Regarding the timely delivery of messages, the *message transmission interval* has to be considered. The message transmission interval starts at the *message send instant*, i. e., the instant when the sender starts to send a message, and ends at the *message receive instant*, i. e., the instant when the receiver fully received the message.

2.1.4 Characterization of Information Content

The fundamental difference between the state of the environment and the representation of the state of the environment within the real-time computer system is addressed by the concept of *real-time entities* and *real-time images* [Kop97, KB03].

Real-Time Entity

The dynamics of a real-time application is represented by a set of state variables, or *real-time entities*. A real-time entity represents a property of the environment that

Attribute	Explanation	Antonym
valid	A message is *valid* if its checksum and contents are in agreement.	invalid
checked	A message is *checked at source* (or, in short, *checked*) if it passes the output assertion.	not checked
permitted	A message is *permitted* with respect to a receiver if it passes the input assertion of that receiver. The input assertion should verify, at least, that the message is *valid*.	not permitted
timely	A message is *timely* if it is in agreement with the temporal specification.	untimely
value-correct	A message is *value-correct* if it is in agreement with the value specification.	not value-correct
correct	A message is correct if it is both timely and value-correct.	incorrect
insidious	A message is insidious if it is permitted but incorrect.	not insidious

Table 2.1: Message classification [JKK+02]

is relevant to the real-time application, e. g., the state of a valve that controls the flow of liquid in a pipe. A real-time entity has static attributes, like e. g., the name or a maximum update rate, that do not change over time and dynamic attributes, like e. g., the value set or the rate of change at a particular instant, that change as time progresses.

A real-time entity can have a discrete value set (*discrete real-time entity*) or a continuous value set (*continuous real-time entity*). The state of a discrete real-time entity is undefined during a state change and can thus be captured only during stable intervals, i. e., during intervals in which the state does not change. An example for a discrete real-time entity that is given in [Kop97] is a garage door with the defined states "open" and "closed" and many undefined intermediate states during a state change from "open" to "closed" or vice versa. Contrary to discrete real-time entities, the value set of a continuous real-time entity is always defined.

State Observation of Real-Time Images

A real-time image is the representation of a real-time entity within the real-time computer system at a particular instant, e. g., the digital representation of the current state of a valve. A real-time image is acquired by a *state observation* that captures the state of a real-time entity at the instant t_{obs}. Thus, a state observation is a triple, consisting of (a) the name of the real-time entity that is observed, (b) the captured value of the real-time entity, and (c) the time of observation (t_{obs}):

$$State\ Observation =< Name, Value, t_{obs} >$$

A discrete real-time entity can only be observed during intervals in which the

state of the real-time entity is not changing. A continuous real-time entity can be observed at any instant.

Temporal Accuracy

A real-time image is invalidated by the progression of real-time and thus associated with a *temporal accuracy interval* d_{acc} which is determined by the physical properties of the real-time entity. A real-time image is temporally accurate at instant t when the following condition holds:

$$|t - t_{obs}| < d_{acc}$$

For instance, the temporal accuracy interval of the state of a valve (real-time entity) could be 10 msec for a given application. This would imply that the observed state of the valve (real-time image) is temporally accurate if it has been observed within the last 10 msec, otherwise it is not temporally accurate.

In contrast to a real-time image, a state observation provides a statement regarding the value of a given real-time entity at a certain instant and thus remains valid forever.

Event Observation

An *event observation* captures the occurrence of an event, i. e., a significant change of state of the observed entity, and can be represented by a triple, consisting of (a) the name of the observed event, (b) the attributes of the event, and (c) the time at which the observed event occurred (t_{event}):

$$Event\ Observation =< Name, Attributes, t_{event} >$$

State versus Event Messages

The interaction with the environment of a real-time computer system and the real-time computer system can either be accomplished by periodic (time-triggered) exchange of *state messages*, i. e., messages that contain state observations or by non-periodic (event-triggered) *event messages*, i. e., messages that contain event observations.

2.1.5 Distributed Systems

A distributed system is a collection of independent computers that appears to its user as a single coherent system [TS01]. In a distributed system, *(system) components*, i. e., hardware-software units with behavior and state, are located at networked computers and *communicate and coordinate their actions only by message passing* [CDK05].

Properties of Distributed Systems

Several properties are related to distributed systems and these properties have a significant influence on a distributed system to be useful for a certain purpose. In the following paragraphs, a briefly outline of these properties is given.

Heterogeneity: A distributed system can be built-up by a set of heterogeneous elements with respect to the employed hardware (e. g., different instruction sets or data representations), the network (e. g., different communication protocols), and the software (e. g., different operating systems). To cope with heterogeneity, a common approach is to introduce a vertical (middleware) layer that provides a programming abstraction and masks the heterogeneity of the underlying technologies.

Openness: The openness of a distributed computer system determines whether the system can be extended by new resource-sharing services. Openness is achieved by publishing the key interfaces of the system in such a way that it can be constructed from heterogeneous hardware and software from different vendors.

Composability: The integration of subsystems to cooperate in a distributed system requires that properties that have been established at subsystem level are not invalidated by the integration of these subsystems. Composability with respect to a certain property of a subsystem means that this property is still valid even after integrating the subsystem into the distributed system, i. e., the system properties follow from the subsystem properties [Kop97].

Scalability: A distributed system that is in use over many years or even decades must be open to changes and must not constrain the extensibility of the system in terms of its processing and communication capacity. A system is said to be scalable if it remains effective, even when there is a significant increase in the number of resources and the number of users [CDK05]. It is important to mention that in case the scale of a scalable system increases, the system and the application software does not need to be changed.

Dependability: The dependability of a (distributed) computer system can be defined as the *trustworthiness* of this computer system *such that reliance can justifiably be placed on the service it delivers* [Lap92]. The attributes of dependability are *availability*, i. e., *the readiness for correct service*, *reliability*, i. e., *the continuity of correct service*, *safety*, i. e., *the absence of catastrophic consequences on the environment*, *integrity*, i. e., *the absence of improper system alterations*, and *maintainability*, i. e., *the ability to undergo modifications and repairs* [ALRL04].

Security: The attributes of security are *availability*, *integrity*, and *confidentiality*, i. e., *the absence of unauthorized disclosure of information* [ALRL04].

Concurrency: When several processes exist on a single computer, these processes are said to be executed concurrently. If a computer has more than one processor, these processes can be executed simultaneously (in parallel), otherwise

the execution of these processes must be interleaved. In a distributed system, many computers exist – thus parallel execution naturally exists in such a system. It follows, that synchronization of operations is vital in order to preserve data consistency among different subsystems of a distributed system.

Transparency: Transparency of a distributed system can be defined as *the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components* [CDK05].

Advantages of Distributed Systems

From the view of the environment of a real-time system it makes little difference whether the real-time computer system is set up as a distributed or as a monolithic system as long as the load- and the fault-hypotheses are fulfilled. Only the physical interface, i. e., the behavior of the transducers interacting with the environment are relevant to the environment. However, for the implementation of a hard real-time computer system, the distributed approach is favored for several reasons [Kop97]:

- Using a distributed approach it is possible to establish the property of composability by exactly specifying the message interface of all participating network nodes in the value and the time domain. Therefore, a distributed system offers a constructive approach for building and validating systems built out of heterogeneous subsystems (probably originating from different vendors).

- A distributed system can be designed to be fully scalable. In order to increase the processing power of a distributed system, new nodes can be added within the capacity of the communication channel. In case the capacity of the communication channel is not sufficient, a new cluster can be added and connected to the system via a gateway node.

- Hard real-time systems require a very high system reliability. In the literature a mean-time-to-failure of better than 10^9 hours for ultra high dependable systems is mentioned [SWH95]. Such a high system reliability can only be achieved with a distributed solution because single hardware units have failure rates that are lower by orders of magnitude. Only with a distributed approach it is possible to define physically separated error containment regions and thus to achieve fault-tolerance by replicating nodes.

Distributed real-time systems have been adopted by automotive and aerospace industries. For instance, in modern cars up to 80 controllers are connected by up to five different bus systems [Bro05]. Recent trends aim at integrating application subsystems by different vendors into a single distributed system architecture. Thereby, several initiatives and research projects (e. g., AUTomotive Open System ARchitecture (AUTOSAR) [AUT06b], Integrated Modular Avionics (IMA) [ARI91],

Dependable Embedded COmponents and Systems (DECOS) [OPHES06]) in the automotive, avionic and related domains are concerned with a systematic, domain oriented process to bundle different application subsystems within the same hardware. These approaches target at increased interoperability, a reduction of the number of Electronic Control Units (ECUs), cables and connectors, and an increase in reliability of the overall system.

Structure of a Distributed Real-Time Computer System

A distributed real-time computer system is composed of smaller entities. For the description of a real-time computer system and its entities we use a terminology that is widely adopted for distributed real-time systems and employed for the TTA [KB03].

The basic building block of a distributed real-time computer system is a *cluster* that can interact with other clusters via a so-called *Gateway Interface (GWI)* [Krü97]. Physically, a GWI is set up by a dedicated gateway node that is part of both clusters and that exchanges messages between these clusters. As depicted in figure 2.3, a real-time computer system can consist of several clusters (*multi-cluster system*). However, a real-time computer system can also comprise of only a single cluster (*single-cluster system*).

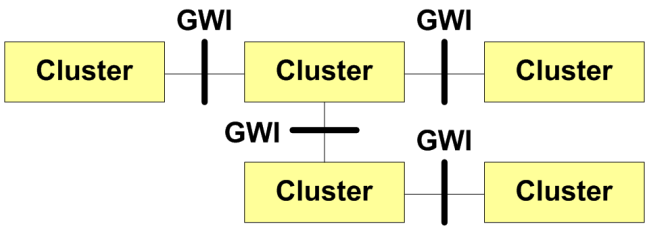

Figure 2.3: Multi-cluster system

Each cluster consists of a set of *nodes* that communicate across a message based interface, i. e., the Logical Line Interface (LLI) (refer to figure 2.4). A node is a self contained computer with its own hardware (e. g., processor, memory) and software (e. g., application programs, operating system), which performs a set of well-defined functions within the real-time computer system [Kop97].

Each node consists of a *host subsystem* executing the node local application and a *communication subsystem* performing the exchange of messages between the respective node and the other nodes of the cluster via the LLI (refer to figure 2.5). The host subsystem and the communication subsystem can be implemented on separate hardware units like for instance in realizations of the Time-Triggered Protocol (TTP) [TTA02] or FlexRay [FX05] in order to enable temporal decoupling of the execution of application tasks and message transmission. For low-cost fieldbus applications, the host subsystem and the communication subsystem can be implemented on the same hardware, i. e., the same micro controller, like for example in TTP/A [OMG03] or Local Interconnect Network (LIN) [Wen02].

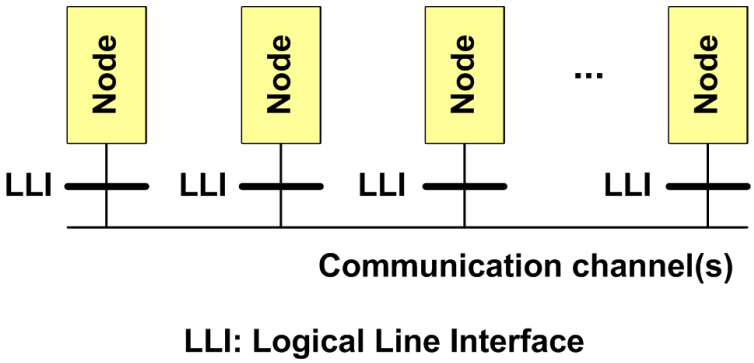

Figure 2.4: Cluster

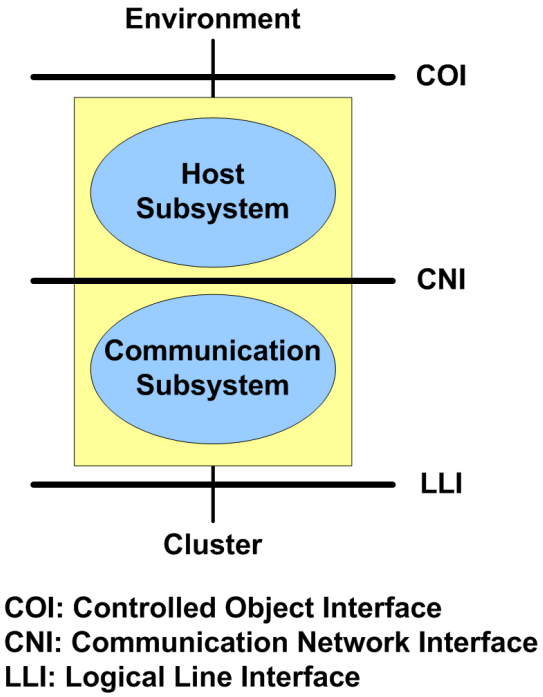

Figure 2.5: Node

The interaction between the host subsystem and the communication subsystem takes place via the *Communication Network Interface (CNI)*. A node operates on the environment – particularly on the controlled object – via the *Controlled Object Interface (COI)* [KFMN99]. Such interaction is established via transducers that are connected to the node. From the perspective of the environment, the COI hides details of the real-time computer system and vice versa (refer to figure 2.6).

For the interaction of a distributed computer system with its environment, only the properties that are defined at the COI are relevant. For an actual physical process like e. g., the control of liquid flow in a pipe, it makes no difference, whether the transducers that act upon the physical process are controlled by one or multiple nodes of a distributed real-time system as long as the properties of the COI are fulfilled.

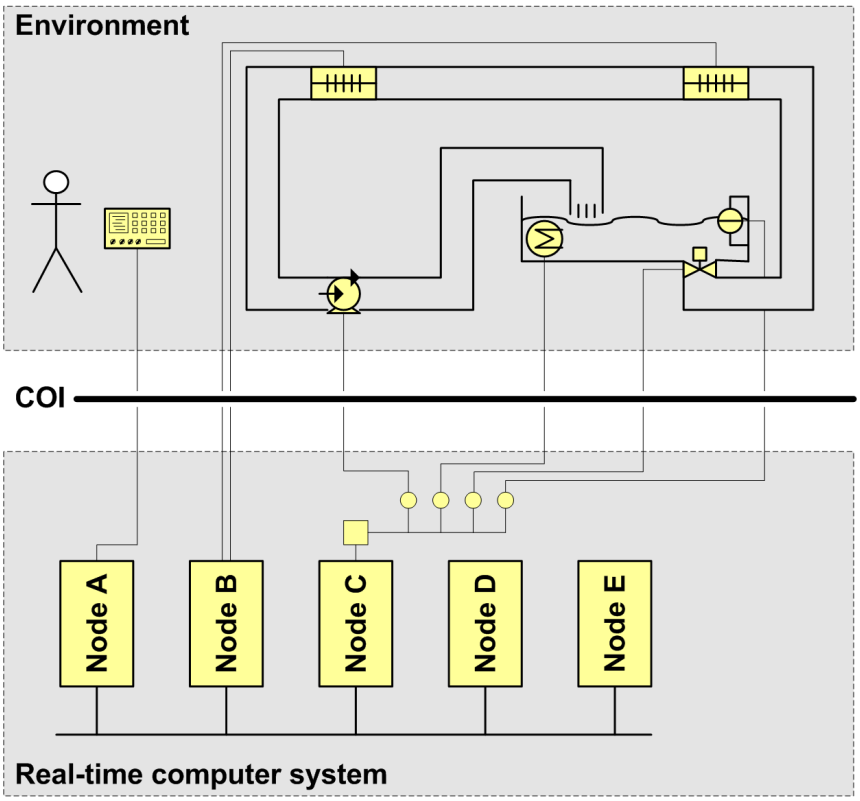

Figure 2.6: Interaction of a real-time computer system with the environment

2.2 Interfaces

According to [JKK+02] an *interface* is a *point of interaction between a system and its environment* where the environment is *everything other than the system.* In [Mil98] the term *communication interface* is regarded as a *structural element* at which the interaction between a (component) system and its environment takes place through the exchange of information.

An interface shall provide understandable abstractions, i. e., essential properties of the interacting component systems [KN97], without neglecting any relevant aspect that could influence these interactions and thereby:

- provide a proper point of interaction between subsystems when partitioning a larger system into smaller subunits in order to reduce the cognitive complexity of the whole system (divide and conquer),
- enable timely information flow between subsystems without introducing additional dependencies between these subsystems,
- hide internal details of the implementation of subsystems, and
- inhibit error propagation among subsystems [Kop99b].

The characterization of an interface between two subsystems of a real-time system can be given by its:

- *syntactic specification* which is concerned with the structure and semantics of the data that crosses the interface, its
- *temporal specification* which is concerned with the temporal constraints that must be satisfied by control signals and data that cross the interface (e. g., instant, phase, and frequency of a given message), and its
- *meta-level specification* that provides a conceptual model of the interface, i. e., the *interface model* that describes the functional intent a user has in mind regarding the information that is passed between the interfacing subsystems [JKK+02, KN97].

An interface can either be an *input interface*, an *output interface*, or both, i. e., a *bi-directional interface*. An input interface consumes information from the environment of a system, an output interface provides information to the environment of a system. An example of an XML interface specification for a distributed fieldbus application is given in [EPS04]. This specification includes the syntactic, the temporal, and the meta-level description of the interfaces between the nodes of the fieldbus application.

2.2.1 Interface State

An interface (partially) represents the state of a system. In the following section, we investigate the notion of state and elaborate on the basic terminology regarding the state of an interface.

In [Pet02], different concepts of state have been identified throughout several engineering disciplines. Basically, these definitions can be divided into the following two different views: First, the state of a system can be regarded to comprise of information determining possible future behavior of this system thereby decoupling the past from the future. This "forward-looking view" is commonly in line with abstract system theory, used by system modelers. Second, the state of a system can be regarded as the total information of a system up to a given instance. This "backward-looking view" is typically favored by system implementors and corresponds to a physical system view.

The state of a distributed real-time system is always a snapshot at a given instant and thus inseparable from the concept of time. The presence of a sparse time base (refer to section 2.1.2) enables a precise characterization of the state of a distributed real-time system because an agreed system wide notion of a certain instant can be established which is a fundamental prerequisite for the definition of the system state.

In [JKK+02], a differentiation between *abstract state*, *stored state* and *declared state* is given. Thereby, the *abstract state* of a system at a given instant is *a notional attribute of the system that is sufficient to determine its potential behavior*. In contrast, the *stored state* of a system at a given instant is *the total information explicitly*

stored by the system (in state variables) up to the given instant. The *declared state* of a system at a given instant is defined as *the value assigned to a declared data structure that can be accessed via an interface and that records all the stored state that is relevant to (i. e., that can influence) the future essential behavior of the system.*

Based on the previous definitions, the state of an interface can be defined as follows [JKK+02]:

Abstract interface state: The abstract interface state at a given instant is defined as the abstract state of a system *as viewed from a particular interface* of this system. The abstract interface state is a *notional attribute* of an interface, that is *sufficient to explain future behavior* of a system *across this interface.*

Stored interface state: The stored interface state at a given instant is defined as the stored state of a system that is *relevant to future behavior at a particular interface.* The stored interface state together with a definition of the system is *sufficient to explain the behavior of the system across this interface.* The stored interface state can be made explicitly available to an interfacing system or it can be hidden behind the interface of the system.

Declared interface state: The declared interface state at a given instant is defined as *the value assigned to a declared data structure that can be accessed via an interface and that records all the stored state that is relevant to (i. e., that can influence) the future essential behavior of the system at the given instant.*

The semantics of the interface state is defined by a *conceptual interface model* which relates the meaning of the chunks of information (i. e., the syntactical interface specification) to the user's conceptual world [KS03].

2.2.2 Interface Types

In order to decrease the cognitive complexity of the system structure, separate interfaces can be defined for unrelated system functions [Kop02]. Usually it makes sense to distinguish between the Real-Time Service (RS) interface, the Diagnostic and Management (DM) interface, and the Configuration and Planning (CP) interface.

Real-Time Service (RS) interface: During operation of a real-time system, the RS interface [EHK01] provides the real-time services (of a real-time system component) to its users. Thus, the RS interface is the most important interface for the service users and should be kept small and understandable. The RS interface is also called *Linking Interface (LIF)* [KS03] because it establishes a link between different system components that interact via the RS interface. We can distinguish between a *Service Providing Linking Interface (SPLIF)*, i. e., the part of the RS interface that offers a certain service to its users and a *Service Requesting Linking Interface (SRLIF)*, i. e., the part of the RS interface that requests services from other system components [KS03].

Diagnostic and Management (DM) interface: The DM interface [JKK+02], also referred to as Diagnostic and Maintenance interface [EHK01], is present during system operation. The purpose of the DM interface is to provide a communication channel to the internals of a system component without disturbing the real-time service across the RS interface. The DM interface can be used for diagnostic purposes, e. g., fault diagnosis, or for setting internal parameters of a component at runtime.

Configuration and Planning (CP) interface: The CP interface [EHK01] is a non-time-critical interface that is required during the integration or reconfiguration phase of a system component. The off-line parameterization of the connection of different system components (e. g., static communication schedule) is handled via the CP interface.

2.2.3 High-Level versus Low-Level Interface Issues

We can distinguish between *high-level interface issues* that are related to semantic, pragmatic and temporal aspects of an interface, and *low-level interface issues* related to transport and representation of information [JKK+02]. Real-time aspects are important on both levels and there is an interdependence between high-level and low-level interface issues regarding real-time aspects, i. e., low-level real-time issues regarding the timing of information transport influence high-level temporal issues.

High-Level Interface Issues

The following high-level interface issues have been identified in [JKK+02]:

Naming: Naming means to associate identifiers to entities within a defined context and to use the identifiers to access the entities [RP93]. Proper naming of entities at an interface has a major influence on the understandability of the interface, i. e., the possibility to establish a correct link between the names and concepts which a human has in mind.

Interaction styles: Component systems can interact in different styles, e. g., by request-reply interaction of a client-server model, by publish/subscribe interaction, by multipeer interaction, or by data passing via a repository.

Dependability attributes: Besides functional attributes, a system may include dependability attributes like timing or delivery guarantees that are part of the interface.

State persistence: If a component system periodically reaches a ground state, i. e., a state where no tasks of the component system are active and no message of the component system is in transit, this ground state can be used as a reintegration point for the component system after a failure. For such reintegration a persistent state is required, i. e., the declared interface state of the component systems linking interface.

Low-Level Interface Issues

Low-level interface issues discussed in [JKK+02] relate to *data representation* of messages that are exchanged via the interface (e. g., byte order), *timing* (i. e., timing of a unidirectional message send and receive operation across the interface), and *flow control* which can be further classified into *explicit flow control* and *implicit flow control*.

In explicit flow control, the receiver controls the speed of information flow by sending an acknowledgment message to the sender upon successful arrival of a message. Thereby, the receiver explicitly signals to the sender its ability to receive further messages. In contrast, implicit flow control involves *a priori* defining at what rate and at which instants messages will be transmitted between sender and receiver. In implicit flow control, the sender implicitly expects that the receiver is able to accept a message as long as the instant of transmitting this message is in accordance with a statically defined schedule. This schedule is available at both, sender and receiver.

2.2.4 Temporal Firewall

A *temporal firewall* is *a unidirectional data-sharing interface with state-data semantics where at least one of the interfacing subsystems accesses the temporal firewall according to an a priori known time-triggered schedule and where at all points in time the information contained in the temporal firewall is temporally accurate for at least d_{acc} time units into the future* [KN97].

To ensure proper information flow between sender and receiver of a time-triggered system (refer to section 2.1.1), the sender (producer) of information must provide temporally accurate real-time images of a real-time entity and must assure that the temporal accuracy of these real-time images is maintained, even immediately before the point of update of the real-time image. The receiver (consumer) of the information must sample the real-time image contained in the temporal firewall and ensure that the accessed information is temporally accurate at its time of use.

For component systems acting as both, sender and receiver (bidirectional communication), the RS interface consists of two separate firewalls, an *input firewall* and an *output firewall*. Temporal firewalls can be used at the CNI, at the COI, and at the GWI of a distributed real-time computer system (refer to section 2.1.5).

A temporal firewall introduces structure into a system by stable properties and thereby simplifies the cognitive complexity of the system. These stable properties constitute important preconditions at the input firewall and postconditions at the output firewall in order to validate a component system. Furthermore, a temporal firewall is free of control signals, i. e., there is no possibility of control-error propagation across a temporal firewall. A data error can only propagate from sender to receiver. In a temporal firewall it is impossible for data error to propagate back from receiver to sender.

2.2.5 Connection System

The cases where two or more real-time component systems are connected, agreement among the RS interfaces of the component systems is required with consideration of their shared set of concepts and their notion of time. In presence of such agreement, i. e., if matching properties of the RS interfaces of the connected components exist, the connection of these interfaces is called *boundary line* [JKK+02]. In the case of a *property mismatch*, i. e., disagreement among the interfaces in at least one of their properties, a *connection system* must be placed between the interfaces of the connected components [JKK+02]. As shown in figure 2.7, a connection system has (at least) two boundary lines in order to connect two systems with property mismatches at their respective interfaces.

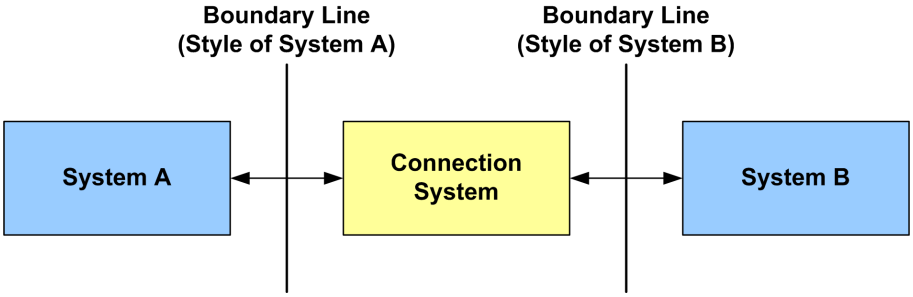

Figure 2.7: Connection system with two boundary lines

A connection system can be regarded as a component system translating interaction requests (resp. responses) from system A into requests (resp. responses) that are understood by system B and vice versa. The duration that is required for performing such a translation must be limited in order to guarantee deterministic real-time behavior of the resulting system, i. e., a system of systems consisting of system A, system B, and the connection system.

2.3 Simulation

Most definitions of *simulation* found within the literature include the concept of a *simulation model* or *model* in short, i. e., a proper abstraction of one or more real-world entities. In [IEE89], a *model* is defined as *an approximation, representation, or idealization of selected aspects of the structure, behavior, operation, or other characteristics of a real-world process, concept, or system.*

The execution of the model for the purpose of conducting experiments with this model is generally referred to as *simulation.* Furthermore, the development of the model may be included to be part of simulation like in [Sha75], where simulation is regarded to be *the process of designing a model of a real system and conducting experiments with this model*, or in [IEE89], where simulation is defined as *the process of developing or using a model* that *behaves or operates like a given system when provided a set of controlled inputs.*

Within the scope of this thesis, we focus on the real-time execution of a model.

Thus, we define *simulation* as the *imitation of the operation of a real-world process or system over time* [BCNN01], i. e., the execution of a model by a *simulator*. A *simulator* is a computer program that is executed on one or more processors and can execute a model to generate its behavior [Clo98].

Overall, the purpose of simulation is to *draw inferences concerning the operation characteristics of the real system* [BCNN01] in order to *understand the behavior of the system or to evaluate strategies for the operation of the system* [Sha75]. With simulation it is possible to abstract from real-time and/or space *in order to point out interactions that would otherwise not be apparent due to their separation in space and/or time in the real system* [Gal99]. For instance, a simulation could show the effects of inflating an airbag during a crash test or a fast chemical reaction in a very slow motion and would allow the user of this simulation to step forward and backward to study each intermediate step of the simulation.

Simulation of a physical process through the execution of a simulation model on a computer involves various concepts of time: First, the physical process itself is time consuming within its real physical environment. For example, the interval "of interest" while observing a chemical reaction could be 115 seconds. We call this time *physical time* since it relates to the physical process. Second, the simulation model must include means to represent the physical time (e. g., with an integer variable representing seconds of the physical time). We call the representation of physical time within the simulation model *simulation time*. Third, execution of the simulation model on a computer is time consuming (e. g., three seconds for the simulation of the chemical reaction on a given computer). We refer to time, measuring execution of the simulation, as the *wall clock time*.

Physical time: The physical time is the (real-)time of the physical process being simulated.

Simulation time: The simulation time represents the physical time within the simulation.

Wall clock time: The wall clock time is the (real-)time of the computer executing the simulation model.

In a (simple) computer simulation that involves only one computer and does not require real-time interaction with an external system during the simulation, it is possible to abstract from physical time and to manipulate the simulation time and thereby accelerate, decelerate or even rewind simulation time. Hence, it is possible to simulate a physical process at a different pace than the predetermined physical time.

However, a simulation may include interaction with an external system, i. e., with parts of the real-world. We will use the term *real-time simulation* to denote a simulation interacting with parts of the real-world. In a real-time simulation, the simulation model is intended to be used as part of a real-world system and it is therefore impossible to abstract from physical time. Hence, in a real-time simulation, physical time,

simulation time, and wall clock time all refer to *real-time* and it is therefore pointless to distinguish between these different notions of time. Real-time proceeds from past to future according to the Newtonian model (refer to section 2.1.2) and may be synchronized to a worldwide standard (e. g., Temps Atomique Internationale (TAI), Universal Time Coordinated (UTC)).

40 years ago, real-time simulation had already been mentioned within the literature [SS66] and since then has meant a simulation that must meet the timing constraints given by the (real-time) behavior of the real-world [Jim05]. A *real-time simulator* must therefore not only interact with its environment in terms of the value domain, i. e., deliver value-correct information, but also in the time domain, i. e., exchange information in agreement with a temporal specification.

In many cases there is a need for fast simulators processing complex simulation models in very short intervals in order to fulfill the temporal requirements of a real-time simulation. To enable fast computing, powerful distributed simulators can be employed. However, real-time simulation must not be confused with the term fast simulation. A real-time simulator must consume its input and produce its output at certain instants. Both, early and late consumption/provision of this input/output would invalidate the simulation run.

It should be mentioned that there is a difference in the taxonomy of *emulation* and *simulation*. In [IEE89], *emulate* is defined, *to represent a system by a model that accepts the same inputs and produces the same outputs as the system represented*. An example of an emulator would be the imitation of a game console on a desktop computer. A system A emulates another system B, if the behavior (i. e., the traces of activity at the systems interface) of system A are exactly the same as for system B – both in the functional and in the temporal domain. The aim of emulation is to resemble system boundaries of the emulated system. To the contrary, system A simulates another system B, if it executes a model that is a mathematical approximation (best representation) of the internal mechanisms of system B. Thus, simulation is concerned with interactions in the sphere of the simulated system.

Drawing a clear distinction between simulation and emulation, related to HiL, can be difficult. It is for example unclear whether the device mimicing the environment of the SUT, i. e., producing input information for the SUT and consuming output information from the SUT, should be called environment emulator (because it *accepts the same inputs and produces the same outputs as the system represented* [IEE89]) or environment simulator (because it is a *device that performs simulation* [IEE89] of the environment).

In line with most literature, the term simulation will preferentially be used throughout this thesis. The term emulation will only be used in cases where the term simulation would definitely be inappropriate.

2.3.1 Classification of Simulation

Within this section, a classification of different simulation approaches is discussed with respect to their *type*, *distribution* and *domain* as outlined in [Gal99].

Simulation Type: We can distinguish between *discrete* and *continuous* simulations. In a discrete simulation, the state variables of the simulation model change solely at discrete instants. In a continuous simulation, these state variables change continuously over time [BCNN01]. A discrete simulation is based on a sparse time base, whereas a continuous simulation is based on a dense time base (refer to section 2.1.2). A discrete simulation can be mathematically described by difference equations, while a continuous simulation can be mathematically described by differential equations.

Discrete simulations can be further classified into *discrete time* and *discrete event* simulations [Gal99]. A simulator that is based on a discrete time simulation, increments an internal timer tick by tick and triggers events as soon as their activation time equals the value of the internal timer. In contrast to a discrete time simulation, where simulation time advances from tick to tick, in a discrete event simulation the simulation time advances progressively from event time to event time.

Simulation Distribution: We distinguish between *central* (monolithic) and *distributed simulations*. In a central simulation a single processor is used to sequentially carry out the execution of the simulation model whereas in a distributed simulation the simulation model is distributed over multiple processors that are able to execute parts of the simulation model in parallel.

The main advantage of a distributed simulation is in the performance gain of the distributed simulation compared to a central simulation. However, cognitive complexity during setup of a distributed simulation is in general higher than cognitive complexity of a central simulation. An example for an approach to distributed real-time simulation, tackling the effective decomposition of a (distributed) simulation model, is presented by the distributed time-triggered simulation scheme [Kim04].

Simulation Domain: We can distinguish between *network simulation, protocol simulation, environmental simulation, and cluster simulation*:

- Network simulation is concerned with the simulation of properties of a given communication network, e. g., performance issues or topology.
- Protocol simulation targets at the evaluation of features of the used communication protocol. An approach to simulate the real-time communication protocol TTP can be found in [Gri01]. Furthermore, multi cluster clock synchronization in a TTP network has been investigated through simulation in [Han04].
- Environmental simulation deals with the simulation of the physical environment of a real-time system. In [Sch95] a distributed discrete time environment simulator for a time-triggered real-time system is presented.
- Cluster simulation [Gal99], sometimes labeled as the "rest-of-the-bus simulation" [FRB97], refers to an approach, where a dedicated device, the so-called cluster simulator, emulates the behavior of one or more nodes of a distributed real-time system.

Simulation Software

The selection of appropriate simulation software plays a crucial role in the development of a simulation. Many simulation languages, containing elements able to generate simulation models, exist. A comprehensive analysis of simulation languages can be found in [Gal99] and in [BCNN01].

A wide range of simulation packages extending existing simulation languages are available. An overview of commercial simulation packages is presented in [Ell00]. An attempt for an open source solution is targeted by the OpenSML project which is based on the Simulation Modeling Language (SML) [Kil01, Wie02].

As an example for a specialized solution for the simulation of distributed real-time systems it is worth mentioning TrueTime [HCÅ03, AHC05], which is a toolbox based on MATLAB/Simulink. TrueTime has been developed since 1999, is available as freeware, and offers several features for the simulation of real-time kernels and network protocols (e. g., support for wireless networks in its latest version [AHCÅ05]). Overall, a comprehensive investigation on a large number of simulation languages and simulation packages is necessary to identify which simulation software is most suitable for a given purpose. In order to aid the selection of adequate simulation software amongst a large range of available solutions, selection methodologies are investigated in [NP99].

2.3.2 HiL Simulation

HiL simulation is a technique where parts of the real system are replaced by a simulation. Thereby, real and simulated components of a (real-time) system are combined into an operational configuration in order to simulate and test the dynamic behavior of the real components, i. e., the SUT [Cra05]. The SUT is implemented on actual hardware while the remaining (simulated) systems, involved in the simulation and testing procedure, are represented by a real-time simulation [Jim05]. The output provided by the simulation is converted (e. g., by D/A modules) and supplied as inputs to the SUT. The output of the SUT is converted (e. g., by A/D modules) and supplied as input to the simulation [Woj99].

HiL simulation is regarded to be a standard method for testing an embedded controller before its deployment [NI03]. Compared to SiL simulation, HiL simulation offers increased realism with respect to the behavior of the simulation because access to hardware features not available in an SiL simulation is provided.

Chapter 3 provides a more detailed investigation on HiL simulation and places further emphasis on HiL simulation of distributed real-time systems.

2.3.3 Validation and Verification

HiL simulation supports the validation of a real-time system, i. e., the *confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled [ISO 9000]* [Vee06]. Hence,

validation *checks that the product design satisfies or fits the intended usage*, i. e., the right product has been built [Wik07c]. Validation is carried out by *dynamic testing* in which the SUT is executed *under specified conditions*, the results of this execution are *observed or recorded*, and an *evaluation is made of some aspects of the system* [IEE90]. We can distinguish between *black box testing* and *white box testing*. Black box testing relates to testing without reference to the internal structure of the system, whereas white box testing is based on an analysis of the internal structure of the system [Vee06].

Verification is the *confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled (ISO 9000)* [Vee06]. Thus, verification *ensures that the final product satisfies or matches the original design*, i. e., that the product has been built correctly (according to its specification) [Wik07c]. Verification can be done by static testing, where the *sanity of a code, algorithm, or document* is checked by syntax checking, manually reading of the code or document, reviews, inspections and walkthroughs [Wik07b].

In order to validate a given system (i. e., to perform dynamic tests on this system), a set of *test cases* is required. Each *test case* includes *input values*, *execution preconditions*, *expected results*, and *execution postconditions* [Vee06]. The *test coverage*, expressed as a percentage, indicates to which degree an anticipated real-world scenario is covered by the corresponding test scenario [Sch92]. Although it is desirable to maximize the test coverage (ideally up to 100 percent), this is often aggravated by limitations like the difficulty to provide realistic test data or to have omniscient knowledge about a systems environment.

It is usually impractical or even impossible to test a system using every possible input combination. Thus, potential test input patterns have to be selected based on criteria of the actual test process. Subsequently, test input congruent to these input patterns has to be generated [Pal00]. For test input generation we can either employ a *probabilistic test technique*, where test sets are based on a random distribution of the input field, or a *deterministic test technique*, where test sets are determined through a selective choice [AFI+00].

Validating a Distributed Real-time System

In a distributed system, multiple loci of control exist because of physically separate processors executing programs independently of each other. Furthermore, a distributed real-time system must not only be congruent to its functional, but also to its temporal specification. In [Sch92] a comprehensive survey on the influence of distributedness and the influence of real-time related to testing of distributed real-time systems can be found. In general, several peculiarities occur, like increased cognitive complexity, accompanied by difficulties of observation, monitoring and control of the system during a test run.

Several issues mentioned within the literature need to be tackled in order to adequately validate a distributed real-time system. Of particular relevance are the properties of *reproducibility*, *representativity*, and *observability* [Sch92, Kop97, Pal00].

Reproducibility (Controllability): The property of reproducibility or controllability of a certain test run is fulfilled if several independent executions of the test run consistently produce the same test results. In order to ensure reproducibility, it may be necessary to (partially) manually control system execution during a test run.

Representativity (Test Coverage): The selection of test input patterns, the design of test cases and actual test execution must be representative of expected lifetime operational conditions. Failing to produce a test coverage of 100 percent suggests the possibility that some *rare event* has not been considered which in turn leads to inappropriate system behavior.

Observability (Probe Effect): All inputs and outputs of a system must be available to the test engineer. In the case where a system imposes different behavior as a consequence of being observed during the test, a so-called *probe effect* [Gai86] has occurred. Preventing the occurrence of probe effects is an important prerequisite in the design of a test environment enabling representative test runs.

In [Kop97] several techniques promoting a better system structure and thereby facilitating system testing, have been proposed and are as follows:

- partitioning the system into composable subsystems which can then each be tested in isolation,
- establishment of a static temporal control structure which is independent of input data and can be tested in isolation,
- reduction of the size of input space by using a sparse time base,
- output of the internal state of a node while it is in ground state in order to improve observability, and
- provision of replica determinism in software in order to improve reproducibility of test runs.

Time-triggered systems allow for the above mentioned techniques because of their architectural properties that lead to deterministic behavior. Because of these architectural properties, time-triggered systems are in general easier to test than event-triggered systems [Kop97].

X-In-the-loop Testing

Apart from HiL simulation, the literature has covered some more 'in-the-loop' techniques. These techniques are generally referred to as *X-in-the-loop* [BSS05].

Model-in-the-Loop (MiL): MiL denotes a technique where the model of a specified function is checked with a simulated system model. Thus, both, the system as well as the environment exist as a model. The application of MiL is relevant in cases where the embedded software development follows a model based approach [LTH03, ZSL+04], and where model based tests are part of the integrated development environment.

Software-in-the-Loop (SiL): Using SiL, executable code is tested in a simulated (software) environment. Executable code may be automatically generated from a model. Furthermore, the test environment from MiL tests may be appropriate to subsequent SiL tests.

Processor-in-the-Loop (PiL): The term PiL [ZLD+04] is used for a technique where the processor of the target hardware is used for testing the executable code. This processor is mounted on a specific evaluation hardware board. PiL bridges the gap between SiL and HiL.

Hardware-in-the-Loop (HiL): With HiL, executable code is tested on the target hardware whilst the environment of the target hardware is simulated. The target hardware used for an HiL simulation can either comprise a single node or a distributed system, consisting of several nodes.

In contrast to X-in-the-loop, *rapid prototyping* is an approach where executable code of a system is tested on prototype hardware. This prototype hardware does not necessarily contain the same hardware elements (e. g., processor, memory, I/O) as the target system. Furthermore, this prototype hardware is tested in the real system environment, e. g., in a real car, instead of a simulated environment.

It should be mentioned that the application of X-in-the-loop simulation techniques for the validation of a system requires that the applied simulation model is sufficiently similar to the real system behavior. Thus, the validation of the simulation model is crucial in order to produce a model that closely resembles true system behavior, thereby maximizing credibility of the model with respect to decision makers throughout the system validation process [BCNN01].

Certification of Real-Time Systems

Certification is the *legal recognition by the certification authority that a product, service, organization or person complies with the requirements*, and includes *activities of technically checking the product, service, organization or person* as well as *a formal recognition of compliance with the applicable regulations* [IMA05]. Certification requires the development of a *safety case* which is *a documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment* [BB98]. Hard real-time systems that are employed in safety-critical domains, like for instance the aerospace, automotive, or space domains, must usually be certified to comply to an accredited standard.

Currently, the Radio Technical Commission for Aeronautics (RTCA) standards DO178-B [RTC92] and RTCA DO254 [RTC00] are relevant for the certification of

airborne systems in order to get approval by the Federal Aviation Administration (FAA). In the automotive area there are a couple of standards and guidelines, such as the Motor Industry Software Reliability Association (MISRA) guidelines [MIS04] or concrete instantiations of the (generic) International Electrotechnical Commission (IEC) 61508 standard [IEC05]. The objectives of these standards share, despite substantial differences, a principle support towards the development of safe software/hardware systems by providing a framework defining the so called "best practices" for the development of such systems.

The potential role of HiL simulation with respect to a certification procedure relates to the software testing process. The DO178-B standard specifies the software testing process in detail and points out that the testing process targets at two objectives [RTC92]: first it shall be demonstrated that the software satisfies its requirements, second it shall be demonstrated that unacceptable errors have been removed. In order to reach these objectives, it may be necessary to involve a *high fidelity simulation of the target computer environment* [RTC92].

However, in case a simulator (resp. emulator) is used throughout the test process, this simulator (resp. emulator) must be qualified and differences between the simulation and the real environment must be considered. Thereby, the test coverage of a test run with an HiL simulation directly relates to the model coverage of the simulation model. In [Bal97] several guidelines for the verification and validation of simulation models are presented that aim at giving accreditation of the accuracy of a simulation model.

2.4 Smart Transducer (ST)

As stated earlier, the interaction between a real-time computer system with its environment via the COI is established via *transducers*. A transducer is a device that converts energy from one form to another in order to measure a physical quantity or to transfer information [ATI01]. Thus, the term transducer subsumes the terms *sensor* and *actuator*, i. e., any transducer in operation is functioning at any given moment either as a sensor or as an actuator [BV99].

A sensor is a device that captures a real-time entity of the controlled object, e. g., thermal, electromagnetic, mechanical, chemical, or optical properties of the controlled object. Thereby, a sensor responds to a physical stimulus by producing a signal, usually electrical [ATI01]. In contrast, an actuator operates on a real-time entity by imposing a condition on the real-time entity, i. e., an actuator produces a physical stimulus.

A *Smart Transducer (ST)* is the integration of an analog or digital transducer element with a local microcontroller and the respective interface circuitry [KHE01]. The microcontroller is used to transform raw transducer signals to a digital representation and vice versa. Furthermore, it is possible to perform checks on the physical transducers as well as a calibration of the transducer signal. An ST includes a message based communication interface that is used to exchange the digital representation of transducer signals with its users.

Figure 2.8 depicts an ST consisting of an Atmel 4433 microcontroller functioning as the processing and communication unit and a Sharp infrared (IR) distance sensor functioning as the physical transducer element.

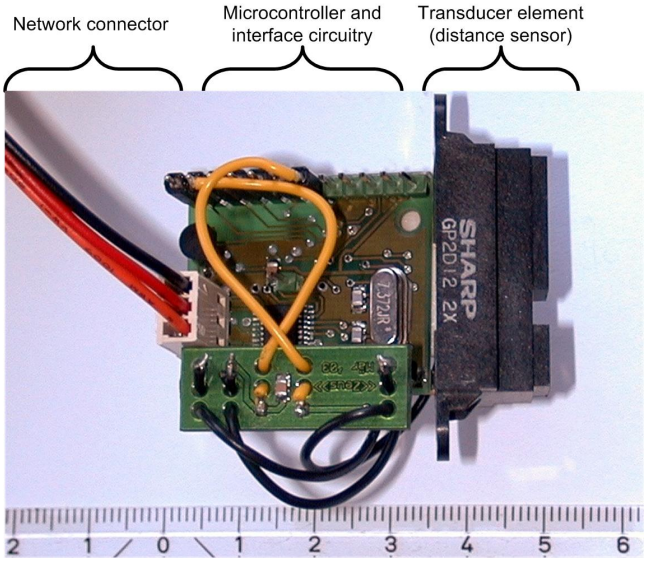

Figure 2.8: Smart Transducer (Atmel 4433 microcontroller and Sharp IR distance sensor, scale in cm)

2.4.1 Smart Transducer Interface (STI)

A (digital) Smart Transducer Interface (STI) supports plug-and-play configuration, monitoring, and communication of digital transducer data via a shared communication channel. However, it keeps the internal complexity of ST hardware and software, as well as internal sensor failure modes hidden from the user [Kop99a].

The interconnection of ST nodes is enabled through a *fieldbus*. The aim of fieldbus technology has been to replace the 4-20mA analogue standard being in use since the 1970s. Using the 4-20mA standard, a unified way for interfacing a transducer exists. Compared to the 4-20mA standard, a fieldbus targets at reduced wiring and lining, easier debugging and maintenance, increased performance by providing a digital interface and the possibility to build networks of (smart) transducers. At present, several different fieldbuses are available. Amongst many others, renowned fieldbus examples are the Controller Area Network (CAN) and the Local Interconnect Network (LIN) within the automotive domain, as well as the Process Field Bus (PROFIBUS) within the factory automation domain.

From the perspective of an end-user, the availability of one world wide accepted fieldbus standard would be desirable. However, suppliers would fail to benefit from such a standard as they would loose their competitive edge by missing out a differentiation across the market [Pin95]. Nevertheless, a couple of fieldbus solutions have been employed successfully in industry, several standards exist and substantial effort has been made to unite these standards. A clear move towards a single interoperable

fieldbus standard has been undertaken by joining forces of two large fieldbus groups, the Interoperable Systems Project and the Flux Information Processus or Factory Instrumentation Protocol to form the Fieldbus Foundation in 1994. The Fieldbus Foundation is a non-profit organization consisting of more than 350 suppliers and end users of process control and manufacturing automation products. The Fieldbus Foundation contributed to fieldbus standards development of the IEC and the Instrumentation Society of America (ISA).

After more than one decade of struggles between various national fieldbus standards, the IEC worked out the IEC 61158 standard in 2000 [Fel02]. The IEC 61158 standard is based on eight existing fieldbus solutions (Foundation Fieldbus, ControlNet, Ethernet/IP, Profibus, SwiftNet, WorldFIP, Interbus, and P-NET). Three years earlier, in September 1997, IEEE 1451.2 was approved. IEEE 1451.2 defines interfaces for an ST, i. e., a digital interface and communication protocol for the connection of transducers and a microcontroller [EP03].

In response to a call for a proposal of a novel STI standard by the Object Management Group (OMG) in December 2000, members of the DSoS project in cooperation with European and US companies and the NextTTA project produced a new standard that is based on the DSoS conceptual model [JKK+02]. This standard has been adopted by the OMG in January 2002 and includes a time-triggered transport service for the interconnection of STs as well as an interface to a Common Object Request Broker Architecture (CORBA) environment [OMG03]. An implementation of the STI including the time-triggered transport service has been realized in the time-triggered fieldbus protocol TTP/A [EO02].

2.4.2 TTP/A – Principles of Operation

A fieldbus that should be used as part of an embedded real-time system must adhere to several requirements [KEM00, EK05]:

Real-time capability: The temporal predictability of periodic real-time communication must be guaranteed, including low latency and minimal jitter. At the same time, high data efficiency must be provided.

Dependability: Although a fieldbus is not designed for the use as the backbone of a safety-related system, it must have an adequate level of dependability and a high error-detection coverage in order to keep maintenance costs low.

On-line diagnostics: It should be possible to parameterize, calibrate, and diagnose an ST node – ideally during runtime and without influencing other ST nodes.

Extensibility: The connection of new ST nodes must be supported in order to allow an integration of new features. Furthermore, support for reconfiguration, exchange, and removal of STs during the lifetime of a system is necessary.

Low complexity and low cost: In order to make the use of the fieldbus economically feasible, low cost microcontroller nodes must be supported through low

protocol complexity and the fieldbus solution must aim at reducing wiring and installation costs.

The OMG STI implementation of the time-triggered protocol TTP/A aims at these requirements. TTP/A supports a two-level design approach, offers three interface types (i. e., an RS interface, an DM interface, and an CP interface as presented in section 2.2.2), and provides automatic configuration of ST nodes [EP03]. The two-level design approach helps in reducing the overall system complexity by separating the transducer implementation from interaction issues with other transducers. The provision of separate interfaces for the RS, DM, and CP aims at a clean separation of functionality in order to achieve a better understandability of the STI. The automatic configuration is supported by so-called cluster configuration descriptions comprising the communication schedule of the cluster, descriptions of participating nodes, and properties for the configuration of the fieldbus (e. g., baud rate).

In the following section, we summarize the key principles of the TTP/A protocol, i. e., the principles of communication, the concepts of a distributed interface file system, and the TTP/A round types, as outlined in [KHE01, EHK01].

Time-Triggered Communication

The TTP/A protocol supports up to 255 nodes all sharing a common data bus. TTP/A is a time-triggered master slave protocol, where a single node acts as the master. The master periodically initiates communication and execution activities by triggering one of eight pre-configured *rounds*. This is done by sending a particular eight bit pattern, a so-called *fireworks byte*, on the idle bus. After that, each node sends and receives messages and executes application tasks according to a predefined round schedule.

The nodes of a TTP/A cluster arbitrate the shared bus according to a *Time Division Multiple Access (TDMA)* scheme, i. e., a node is only allowed to send a message at a given instant and each node has knowledge about the instants when messages from other nodes are to be received. Each message consists of one or more bytes that are transmitted in the Universal Asynchronous Receiver Transmitter (UART) format within a *slot* of a TDMA round.

The common time base required for a TDMA bus access scheme, is provided by the master node. Therefore, the arrival event of the fireworks byte is used for starting a common time-base at each node. Furthermore, synchronization slots during a round can be used for re-synchronization during a round.

Interface File System

All actions within a TTP/A cluster, i. e., communication and task execution, are organized in a distributed file system that is called the *Interface File System (IFS)*. The IFS is structured in a record-oriented format and provides a shared name space for all messages that are exchanged within a TTP/A cluster. Both, the nodes static

configuration and dynamic data elements are included in the IFS. The TTP/A protocol provides a unified mechanism to access each file entry within the IFS.

The IFS allows three file operations, *read*, *write*, and *execute*. Read/write are atomic operations that read/write a record from/into a file of the IFS. Execute initiates the execution of a file of the IFS. Four different file types in the IFS can be distinguished: *read-only documentation files* that contain the documentation of a node, *input-output files* that are used to store data during runtime, *Round Descriptor List (RODL) files*, and *special command files* that contain executable code.

Each TTP/A node contains at least three files, a *RODL file* describing the TDMA schedule of the node, a *configuration file* including at least the name of the node, and a *documentation file* comprising at least the series and serial number of the node. The name of the node is used for addressing a node in a master slave round. The series number defines the type of the node, while the serial number is used to differentiate between nodes of the same type.

Round Types

As mentioned earlier, a TTP/A round is initiated by a fireworks byte which is sent by the master. After the transmission of the fireworks byte, data frames are sent by nodes of the TTP/A cluster congruent to the specification of the round. The round can either be a *multipartner round* or a *master-slave round*.

A multipartner round is used for the periodic exchange of real-time images and consists of a configuration dependent number of slots with an assigned sender node for each slot. The configuration of a multipartner round, i. e., which node of the TTP/A cluster must send data in which slot, is specified in a RODL. The RODL must be configured prior to the initiation of the corresponding multipartner round. Configuration of the RODL is possible at design time and at runtime (during a master-slave round).

A master-slave round is used to establish a connection between the master node and a slave node for operating on the local IFS of the slave node. The slave node is addressed by its name. The operation performed by the slave node is specified in a frame sent by the master node. Thus, a master-slave round can be dynamically initiated during runtime and there is no need for any pre-configured schedule at the slave node.

Typically, multipartner and master slave rounds are interleaved, thereby supporting diagnosis and maintenance services concurrent to real-time services. The implementation of master-slave rounds can be omitted in ultra low cost TTP/A.

2.5 Chapter Summary

A *real-time system* performs actions within *a priori* defined time intervals. Missing a *deadline* within a real-time system leads to degradation or failure of the delivered *service* which can have catastrophic consequences. A *service* is *an operation that*

has a published specification of interface and behavior [AUT06a]. *Behavior* in this context is defined as *traces of activity at system interfaces* which are *sequences of (perhaps timestamped) send and receive operations of the system* [JKK+02]. A *real-time system* consists of the *real-time computer system*, the *controlled object*, and the *operator*. The *controlled object* and the *operator* are regarded as the *environment*. The interaction between a real-time computer system and its environment is established across the Controlled Object Interface (COI). Physically, the COI is realized by *transducers*, i. e., sensors and actuators that transform a physical stimulus into a signal and vice versa.

The aim of an *interface* is to provide understandable abstractions of interacting component systems. An interface can be characterized by its *syntactic, temporal*, and *meta-level specifications*. An interface represents (part of) the *state* of a system. In order to separate unrelated system functions, three types of interfaces can be distinguished in the context of real-time systems: *Real-Time Service (RS) interface*, *Diagnostic and Management (DM) interface*, and *Configuration and Planning (CP) interface*. An important interface concept for time-triggered systems is the concept of a *temporal firewall* that is based on decoupled communication and that prevents control signal propagation across an interface. Translation of interaction requests between different systems is supported by the concept of a *connection system* which resolves property mismatches of these systems.

Simulation is the execution of a *(simulation) model* for the purpose of conducting experiments with the model, thereby imitating the operation of a real-world process or system over time. *Real-time simulation* denotes a simulation where the simulation time needs to be kept synchronous with real-time. *HiL simulation* is a technique where part of a real system, i. e., the *SUT* is implemented on actual hardware while the remaining part is represented by a real-time simulation. HiL simulation is used for *dynamic testing*, i. e., validating an SUT under specified conditions. Key properties for testing an embedded real-time system are: *reproducibility*, *representativity*, and *observability*.

In order to overcome restrictions and to cope with the diversity of different transducers, *fieldbus technology* introduced the concept of a *Smart Transducer (ST)*, i. e., the integration of an analog or digital transducer element with a local microcontroller and the respective interface circuitry. In January 2002, a new standard was adopted by the Object Management Group (OMG) that includes a time-triggered transport service for the interconnection of STs as well as an interface to a CORBA environment. An implementation of this OMG STI standard, including the time-triggered transport service has been realized in the time-triggered fieldbus protocol *TTP/A*.

Chapter 3

HiL Simulation

As briefly outlined in section 2.3, HiL simulation is a technique, where physical parts of a real system are substituted by a simulation. Within this chapter we investigate the basic elements of an HiL simulation as well as the structure of a typical HiL simulation system. Furthermore, we discuss requirements imposed on an HiL simulation and investigate existing HiL simulators.

3.1 Structure and Constituting Elements

An HiL simulation system consists of two major building blocks (refer to figure 3.1):

System-Under-Test (SUT): The SUT is typically part of an embedded real-time computer system. It can be only a single microcontroller, but it can also be a distributed network of communicating nodes. The SUT interacts with its environment, i. e., the operator and the controlled object, across the COI. In case the SUT is part of a larger (distributed) system, it exhibits an LLI enabling the interaction with the surrounding system.

HiL Simulator: The aim of an HiL simulator is to perform experiments with an SUT. Hence, it substitutes the environment or parts of the environment of the SUT. In case the SUT is part of a larger computer system, aiming to interact with this larger computer system, the HiL simulator may provide means for cluster simulation, thereby emulating the LLI of the SUT. In that case, the HiL simulator executes simulation models of both, the environment of the SUT as well as the non-available larger computer system.

3.1.1 Elements of an HiL Simulator

An HiL simulator is constrained by temporal and functional properties of the SUT. The development of the SUT is typically independent of the development of the HiL simulator. Thus, the interfaces of an HiL simulator, relevant to the interaction with

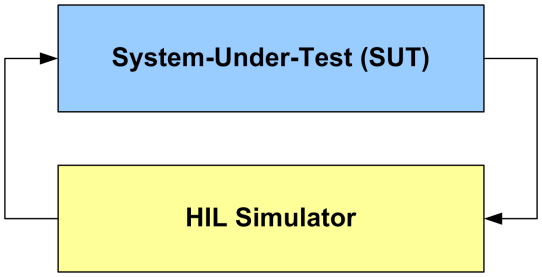

Figure 3.1: Basic elements of an HiL simulation system

the SUT, must be in line with the set of concepts and the notion of time of the SUT's interfaces. In case of property mismatches, a *connection system* (refer to 2.2.5) is required.

An HiL simulator involves the simulation of the environment of the SUT. The simulation of the environment includes a simulation of the controlled object and (if applicable) inputs of an operator. In the case where the SUT is part of a distributed system and the operation of the SUT relies on the presence of non-available parts of this distributed system, an HiL simulator additionally involves the simulation of these non-available parts (cluster). Hence, the basic building blocks of an HiL simulator are (refer to figure 3.2):

Environment simulator: The environment simulator is responsible for the simulation of the environment of the SUT (i. e., the controlled object and the operator).

Cluster simulator: In case the SUT is part of a distributed system, the cluster simulator is responsible for the simulation of missing nodes of this distributed system.

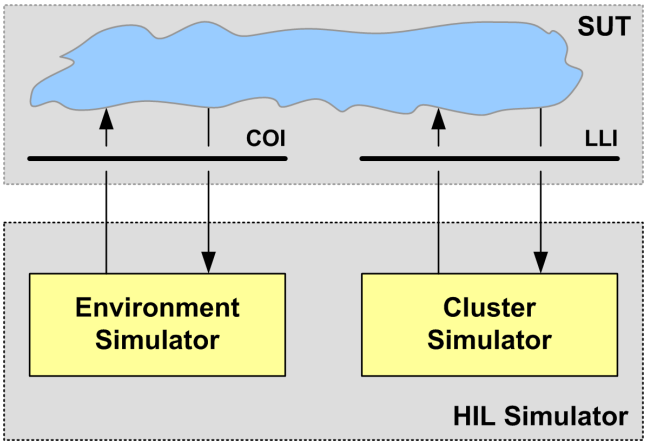

Figure 3.2: Basic building blocks of an HiL simulator

Physically, an environment simulator and a cluster simulator can be built upon the same hardware elements. The purpose of environment simulator and cluster

simulator is to substitute "all missing parts around the SUT" (i. e., the controlled object, the operator, and missing nodes of a distributed system whereof the SUT is a part). For example, in an automotive windscreen wiper application as SUT, the windscreen wiper application would be executed on a node that connects to other nodes (e. g., via CAN). For the HiL simulation we want to solely test the single node that runs the windscreen wiper application. Therefore, we must develop an environmental simulation, e. g., a simulation of the actuator that drives the windscreen wipers and a sensor that captures weather conditions. Furthermore, we must develop a cluster simulation to provide the windscreen wiper node with relevant information, such as car speed or relevant operator commands. In a real system (i. e., after the windscreen wiper node is installed in a car) such information would be provided via other nodes (i. e., CAN messages).

Although environmental simulation and cluster simulation could be combined to establish a single simulation model, we feel that a conceptual separation of these parts is necessary. Such separation is motivated by the following considerations:

- As outlined in section 2.1, there is a clear distinction between the COI and the LLI of a distributed real-time system. Nodes of a real-time computer system interact with other nodes of this system via the LLI. The interaction of the real-time computer system with its environment, i. e., a controlled object and an operator, happens across the COI. In case, one or several nodes of a real-time computer system (i. e., the SUT) are included in an HiL simulation, the environment simulator of this HiL simulation connects to the COI of the SUT and the cluster simulator connects to the LLI of the SUT.
- A conceptual separation of environment simulator and cluster simulator allows a clear distinction between the real-time computer system's internal behavior (even if this system is not yet fully available) and the environment of this real-time computer system. Thus, a clear distinction is made between parts that are under control of the real-time computer system and parts that are not within the sphere of control of the real-time computer system. Although messages received by the SUT from other nodes of the real-time computer system are outside the sphere of control of the SUT, it is usually possible for a system developer to calibrate the general behavior of a real-time computer system. However, it is usually impossible to alter the behavior of the environment of the real-time computer system.

An HiL simulator can either be a centralized system or a distributed system. Typically, one or several interconnected processing units are employed for executing the environmental simulation model and (optionally) the cluster simulation model. Furthermore, dedicated HW boards are used for the interconnection of the processing units of the HiL simulator and the SUT.

As mentioned in [Cra05], an HiL simulator does not only provide interfaces to the SUT. An HiL simulator also comprises interfaces to a human operator and to a postsimulation-analysis platform. The interactive Human Machine Interface (HMI) is used for injecting manual signals, e. g., for fault-injection experiments. The interface

to a postsimulation-analysis platform is required to capture signals that are imposed on the SUT as well as corresponding signals that are received from the SUT.

3.1.2 Open-Loop versus Closed-Loop HiL Simulation

An HiL simulation can be classified into either *open-loop HiL simulation* or *closed-loop HiL simulation*. As depicted in figure 3.3, the difference between these variants is that in an open-loop HiL simulation, the HiL simulator provides predefined simulation data to the SUT while in a closed-loop HiL simulation, the calculation of the simulation data depends on previous output of the SUT.

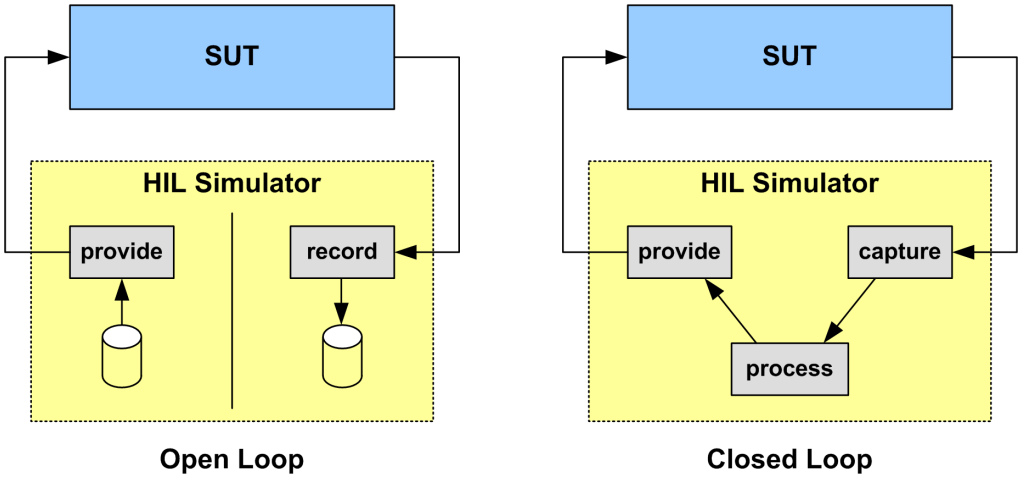

Figure 3.3: Open-loop versus closed-loop HiL simulation

In an open-loop HiL simulation, the calculation of simulation data by the HiL simulator is independent of the previous output data of the SUT. Hence, the dynamic behavior of the SUT cannot be observed. In an open-loop HiL simulation, the required set of predefined input values is typically calculated offline. However, the actual HiL simulation is still operated in real time. The output values of the SUT are recorded for postsimulation analysis. Open-loop HiL simulation is a useful approach for testing low-level input/output drivers, for unit testing, and to some extend for integration testing [NBAR04].

In a closed-loop HiL simulation, previous output of the SUT directly influences the calculation of subsequent input data. Thus, the simulation data must be calculated by the HiL simulator during runtime, i. e., in real time. In a closed-loop HiL simulation, the controller feedback loop is preserved. Hence, a complete environment model with appropriate feedback response by the HiL simulator is required. A closed-loop HiL simulation is applicable for system level testing.

3.1.3 Developing an HiL Simulation

Before developing an HiL simulation, an interface specification of the SUT, including functional and temporal aspects, must be available. Regarding the physical setup of

the HiL simulation, overall, two aspects should be considered [NI03]:

- Hardware able to receive and generate signals to and from the SUT has to be selected. In commercial solutions, dedicated hardware boards are offered that consist of a set of input and output channels, supporting a variety of signals. Refer to section 3.3 for a closer analysis of (commercially) available solutions.
- A suitable processing system carrying out the calculation of the simulation model has to be chosen. The processing system is also used for recording of postsimulation-analysis values and for the interaction with an operator. The processing system can be constructed as a centralized system or as a distributed system.

In case of a closed-loop HiL simulation, a simulation model has to be developed that is then to be executed on the processing system. The possible level of detail of this simulation model depends on the capabilities of the processing system.

In order to display parameters of an HiL simulation during a simulation run, a dedicated HMI computer is used. This HMI computer is usually also used for the development of the simulation model. Furthermore, postsimulation analysis of data that has been recorded during an HiL simulation is typically conducted by the HMI computer.

3.1.4 Multirate HiL Simulation

A classical HiL simulation would involve processing hardware that executes an HiL simulation model and that instruments digital and analog I/O channels. These digital and analog I/O channels are connected to the SUT and send/receive control signals (either analog or digital signals) to/from the SUT. The rate for the recurring execution of simulation steps depends on the temporal requirements of the SUT. The I/O channels that connect to the SUT are updated each time, the simulation model finishes an execution step, i. e., after each step of the simulation model.

HiL simulation with SUTs that exhibit fast feedback loops in the nanosecond range limit the capabilities of the simulation model and require fast hardware solutions (e. g., based on FPGAs). If the processing hardware cannot timely complete execution steps of the simulation model, it is necessary to chose a larger rate. However, increasing the rate lowers the accuracy of simulated signals.

In [GLH06, LGDL06], a dual time step simulation, involving two separate simulators which operate at different rates, is introduced. Thus, a separation between the fast simulation of power converters and the comparably slower simulation of the remaining network is achieved, thereby allowing the latter to execute regular algorithms. The interaction between the two simulators is established by controlled voltage and current sources.

In consensus with the idea of a dual time step simulation, discussed in [GLH06, LGDL06], we propose a *multirate HiL simulation*. A multirate HiL simulation is based on a distributed HiL simulator consisting of *Backend Simulation Components*

(BSCs) and *Frontend Simulation Components (FSCs)* (refer to figure 3.4). Each of these components uses an individual rate for the execution of the simulation model and the interaction with the SUT.

- BSCs execute parts of the simulation model that are independent of low level interconnection details regarding the coupling between the SUT and the HiL simulator.
- FSCs control the interaction between the HiL simulator and the SUT.

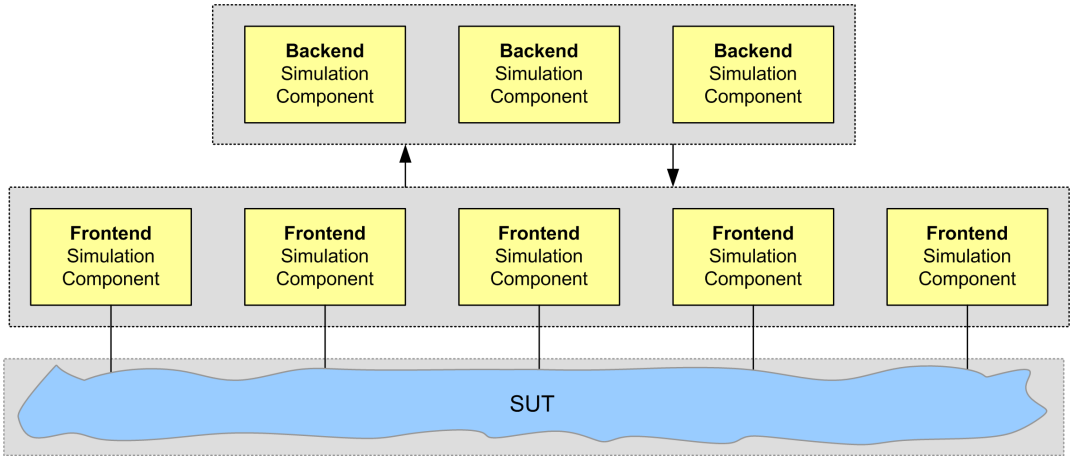

Figure 3.4: Multirate HiL simulation

The example of the automotive windscreen wiper mentioned earlier would for instance require the simulation of weather conditions. These weather conditions would be simulated by BSCs. FSCs would be responsible for the simulation of the step response of sensors and actuators (e. g., read in the Pulse-Width Modulation (PWM) control signal of the windscreen wiper motor controller). Typically, BSCs use a much larger rate than FSCs.

FSCs interact with the SUT via control signals and observations of the behavior of the SUT. A sparse time base (refer to section 2.1.2) is a fundamental prerequisite in order to guarantee consistent ordering of the instants of these interactions. Hence, each component of the multirate HiL simulator (i. e., FSCs and BSCs) must be synchronized to a common (global) sparse time base that is agreed among these components.

3.1.5 Coupling of HiL Simulator and SUT

The interaction between the HiL simulator and the SUT is an essential aspect for any HiL simulation. This interaction highly depends on the functional and temporal requirements of the actual SUT's interfaces. Thus, it is impossible to define a universal solution for the coupling of an HiL simulator and an arbitrary SUT. However, a structured development approach and the availability of generic components that

can be tailored to establish the coupling between an HiL simulator and a specific SUT are desirable to enable a faster development of an HiL simulator.

The proposed distinction between BSCs and FSCs for multirate simulation mentioned above, enables a conceptual separation between those parts of an HiL simulator which are independent from the physical realization of the SUT (i. e., BSCs) and those parts that directly connect to the SUT and therefore depend on the physical realization of the SUT (i. e., FSCs). Besides the ability of supporting different rates through a multirate HiL simulation, the separation between BSCs and FSCs leads to a separation of concerns regarding the coupling of an HiL simulator and the corresponding SUT.

An FSC requires periodic updates of simulation values that are provided by one or more BSCs. Based on these simulation values, the FSC determines the control signal that is to be provided to the SUT. Both, the control logic that calculates the required physical signal based on the simulation value, as well as the physical wiring, are part of the FSC. Thus, a change in the interface specification of the SUT directly affects an FSC, but not necessarily the BSCs as long as the FSC can be provided with all relevant simulation values in time.

Capturing of the SUT's output signals is also in the responsibility of the FSCs. Due to different rates between FSCs and BSCs, it is necessary to pre-process the signals from the SUT at the FSCs. In [MFL+05] a method calculating the average of signals throughout a single simulation step of the respective BSC is proposed.

The availability of separate FSCs in an HiL simulation is particularly advantageous when it comes to incremental testing of a distributed real-time system. Starting with a single node, a stepwise inclusion of nodes of the distributed system in the HiL simulation is required. At each step, the environment model of the real-time system is simulated (by BSCs) and the coupling between this simulation and the actual SUT is established with FSCs. With separate FSCs it is possible to scale the HiL simulation from a small SUT (single node) up to a complete distributed system by adding additional FSCs as required.

In chapter 4 we elaborate on a concept enabling the setup of a scalable, deterministic network of BSCs and FSCs with respect to the temporal dependencies between these components.

3.1.6 An Example

We consider an SUT whose aim is to control a simple circuitry as depicted in figure 3.5. This circuitry consists of a battery, a capacitor (C), two resistors (R_1, R_2) and a switch. If the switch is on, the capacitor is charged and the voltage of the capacitor (U_C) asymptotically approaches the target voltage (U_∞). If the switch is off, the capacitor is discharged through R_2. The instants of opening/closing the switch determine the voltage characteristic of the capacitor ($U_C(t)$).

With this example we can see that considering time as the determining factor of a real-time system is indispensable for achieving desired system behavior. It follows that an HiL simulator must absolutely adhere to the temporal specification of the

SUT's interfaces to enable correct simulation results. Even small deviations with respects to the instants of interaction (between HiL simulator and SUT) might have a big (negative) impact on a particular HiL simulation run.

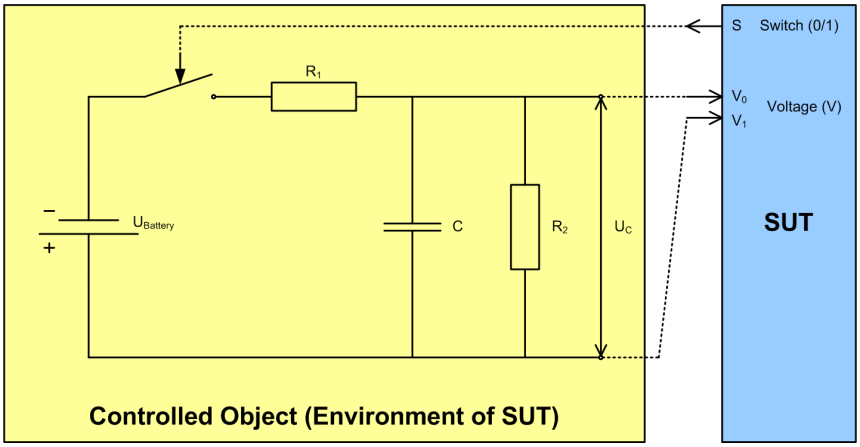

Figure 3.5: Charging and discharging a capacitor (C)

The voltage characteristic of the capacitor during the charging phase (switch on) is given through:

$$U_C(t)_{charging} = U_0 + (U_\infty - U_0) * (1 - e^{-k*t})$$

where U_0 is the voltage of the capacitor at $t = 0$ (i. e., instant where charging starts),

$$U_\infty = \frac{R_2}{R_1 + R_2} * U_{Battery}$$

and

$$k = \frac{R_1 + R_2}{C * R_1 * R_2}.$$

As soon as the switch is turned off, the voltage characteristic of the capacitor is given through:

$$U_C(t)_{discharging} = U_0 * (e^{-k*t})$$

where U_0 is the voltage of the capacitor at $t = 0$ (i. e., instant where discharging starts) and

$$k = \frac{1}{R_2 * C}.$$

Figure 3.6 shows the voltage characteristic ($U_C(t)$) of the capacitor (C) for exemplary values ($U_{Battery} = 12V, R_1 = 3k\Omega, R_2 = 1k\Omega, C = 0,5\mu F$) given that the

capacitor has an initial charge of zero and the switch is turned on for 2ms (charging) and then turned off (discharging).

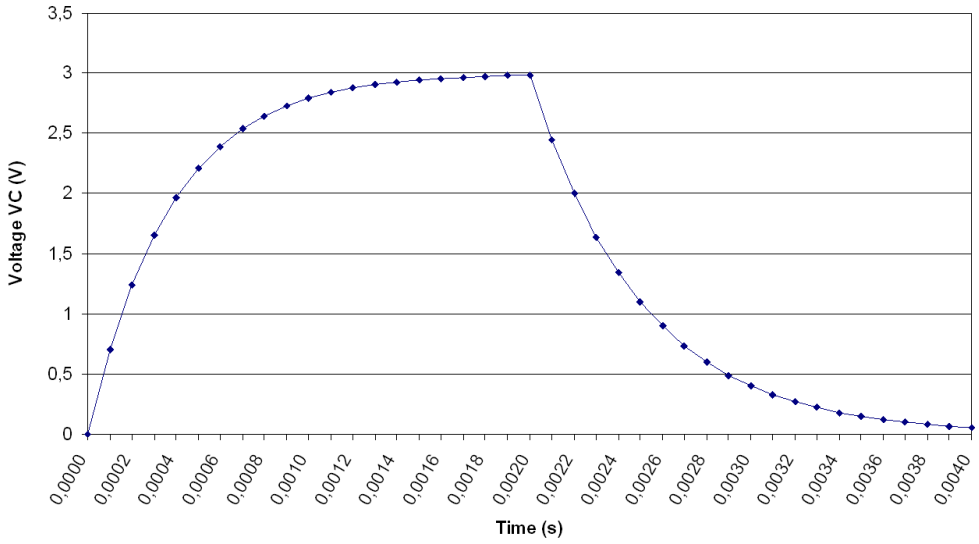

Figure 3.6: Voltage characteristic ($U_C(t)$) of capacitor (C)

The setup of a distributed HiL simulator for this example involves a BSC and two FSCs. The BSC simulates the entity that turns on/off the switch and thereby controls the voltage characteristic of the capacitor (BSC Voltage Characteristic). The FSCs that are connected to the SUT provide the control signal for the switch (FSC Switch) and capture the voltage at the capacitor U_C (FSC Voltage). Figure 3.7 depicts an exemplary setup of the HiL simulator.

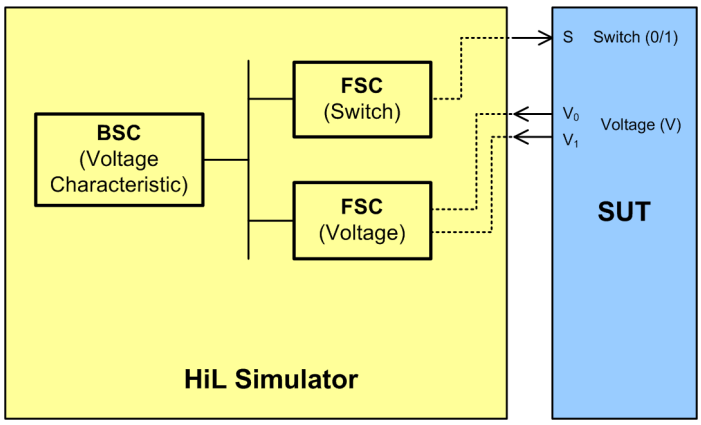

Figure 3.7: Exemplary HiL simulator setup

The rate of the FSCs can be higher than the rate of the BSC (multirate simulation). In any case, the correct timing, i. e., precise instants of interaction between the FSCs and the circuitry, is an essential property of the HiL simulation.

For example, in the case of fast switching rates, it is possible to keep the voltage (U_C) at a certain level, i. e., between two thresholds (high, low). The precision of

the switching instants determine the precision of these thresholds. In figure 3.8, the voltage level of the circuitry is exemplarily given for three different switching rates. Each sample is carried out for 2,4ms. In sample A, the switch is on for 0,2ms, then off for 0,1ms, then on again, etc. In sample B, the switch is equally on/off for 0,2ms. In sample C, the switch is equally on/off for 0,05ms.

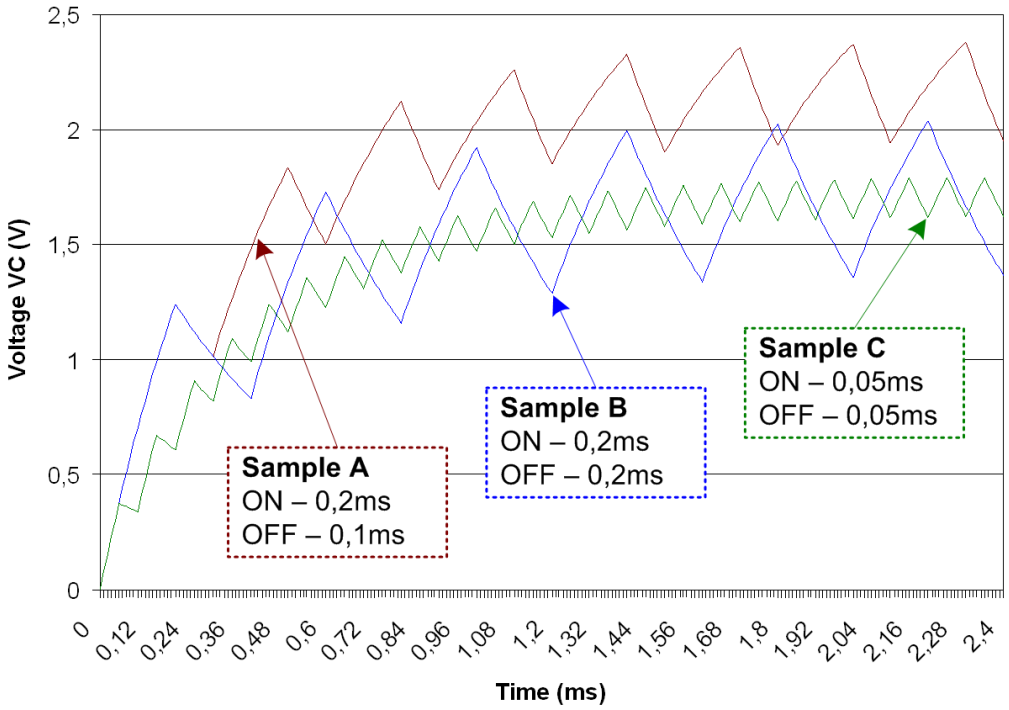

Figure 3.8: Voltage characteristic ($U_C(t)$) of capacitor (C) with different switching times

Different instants of turning the switch on/off in the samples (A,B,C) lead to different voltage levels of the circuitry. It follows that any deviation with respect to the instants of turning the switch on/off (e. g., through temporally inaccurate interaction between the HiL simulator and the SUT) would lead to a deviating voltage level of the capacitor. Hence, temporally inaccurate interaction, which can be caused by sources of indeterminism of the HiL simulator, potentially invalidate the whole simulation run.

3.2 Requirements

In the previous section we discussed the basic constituting elements of an HiL simulation. In practice, the actual setup of an HiL simulation depends on several technical and economic considerations. Thus, before investigating various forms of available solutions of HiL simulation, it is important to first elaborate on requirements imposed on an HiL simulation. Based on these requirements, a proper solution for an HiL simulation can then be either selected (if already available) or has to be newly developed.

In [Ecc00], a characterization of simulator requirements is summarized. Although this characterization focuses on general simulators, most parts can also be applied to the specific field of HiL simulation. Based on the discussion in [Ecc00] regarding the characterization of requirements for general simulators, the following questions have to be answered before starting the development of an HiL simulator:

- What is the *intended use* of the HiL simulation? The purpose of an HiL simulation could be to explore different design alternatives of the SUT. Or the HiL simulation could focus on analyzing specific properties of the actual design of the SUT, e. g., performance, fault-tolerance. However, the purpose of the HiL simulation could also be to present the capabilities of the SUT to (potential) customers.

- What *level of detail* is required for the HiL simulation? The level of detail is given by the *scope* and the *resolution* of the HiL simulation. To define the scope of an HiL simulation, it is necessary to know what part of the real world is to be included in the HiL simulation. The resolution refers to the extend to which the simulation should represent the real world, i. e., the granularity of the HiL simulation?

- What is the *fidelity* of the HiL simulation? Fidelity can be defined as *the accuracy with which a simulator represents the real-world system or systems it represents* [Ecc00]. Fidelity must not be confused with the level of detail of an HiL simulation. A simulation with a high level of detail does not necessarily answer the right questions. A smaller and better understood simulation may be closer to reality as far as the user of the HiL simulation is concerned.

- How can the *data* for an HiL simulation be obtained? Accurate and useful information about the environment of the SUT has to be generated through the execution of a simulation model. Obtaining such data is often a non-trivial process because it requires detailed knowledge or difficult measurements about the physical phenomena relevant within the environment of an SUT.

As soon as these questions have been answered, a closer look at the technical and economic constraints of the desirable solution is necessary. In the following section, we discuss several technical and economic requirements, that must be considered for an HiL simulator aiming to test a real-time system.

3.2.1 Technical Requirements

We have identified four requirements relating to technical properties of the HiL simulator. These are:

- the ability for real-time execution of the simulation model,

- the reproducibility and observability of simulation results,

- the availability of mechanisms for monitoring and data logging, and
- the provision of sufficiently high signal quality for the coupling of the HiL simulator and the SUT.

Real-time Capability

The HiL simulator must be able to execute the simulation model in real time. Thus, it is necessary that the HiL simulator interacts with the SUT congruent to the SUT's temporal interface specification. Thereby, an acceptable trade-off between execution speed and level of detail of the simulation model must be found to achieve real-time capability of the HiL simulator.

We distinguish between the following two approaches for the construction of a real-time HiL simulator:

- The first approach involves a soft real-time system implementing an HiL simulator that is able to detect and report deadline violations during a simulation run. The cause for deadline violations (i. e., temporal requirements of the SUT are not met) can be a temporally unpredictable behavior of the simulation model's execution. In case a deadline violation occurs during a simulation run, the cause of this deadline violation has to be identified and the HiL simulator has to be improved respectively (e. g., performance improvements, or re-design of the communication schedule). After that, the simulation run has to be repeated.
- If it is not acceptable to miss a deadline during the simulation run, the second approach is to use a hard real-time system for the realization of the HiL simulator. With such an approach, real-time behavior can be guaranteed by design. However, the second approach requires more effort during the development of the (hard real-time) HiL simulator. Therefore, it is only useful in cases where either the execution of the SUT is a safety-related process (e. g., because equipment is involved that imposes a risk on the execution of the SUT), or a single simulation run is expensive in terms of computing resources or time and repeating the simulation run is not acceptable from an economic point of view.

Reproducibility and Observability

An HiL simulation must exhibit properties of reproducibility and observability. As outlined in section 2.3, reproducibility means that multiple executions of an HiL simulation always lead to the same results. To achieve reproducibility, the execution of the simulation model by the HiL simulator and the interaction between the HiL simulator and the SUT must be *deterministic*.

From a philosophical point of view, *determinism* is the proposition that every event is causally entailed by an *unbroken chain of prior occurrences* [Wik07a]. The definition of *determinism* as given in [Hoe04] states that *the world is deterministic if*

and only if, given a specified way things are at a time t, the way things go thereafter is fixed as a matter of natural law.

According to the ongoing rich and capacious philosophical reflections on determinism (refer to [Hoe03]) it is virtually impossible to decide whether our natural world is deterministic or not. It is not even possible to know whether this is a decidable question. It follows that a "pragmatic" application of the term determinism requires a rather simplified view of the world and its natural laws.

Although the concept of determinism is indeed subject to controversial philosophical reflections, we think it is at least helpful to start with a domain specific definition of determinism with respect to an HiL simulator. This will help us to point out the intention of this thesis' contribution. In this sense, we argue that an HiL simulator *behaves deterministically* if, given a defined initial state $q_0(t_0)$ of this HiL simulator at $t_0(now)$, a sequence of inputs $a_i(t_i)$ at sparse real-time instants t_i will always produce the same sequence of outputs $b_j(t_j)$ at the same future sparse real-time instants t_j [KESHO07][1].

In order to establish the property of observability, any probe effect (refer to section 2.3) must be avoided. The HiL simulation must not disturb the natural behavior of the SUT, i. e., there must be no difference between the behavior of the SUT during an HiL simulation and the behavior of the SUT within its physical environment. Within the related literature, the term *intrusive* is used in cases where a real-time system is altered for monitoring purposes (e. g., when an additional SW monitoring function is added) [Sma04]. An observable HiL simulation run requires non-intrusive system design.

Monitoring and Data Logging

An HiL simulation must provide mechanisms for monitoring the behavior of the SUT. Furthermore, means for logging the information that has been exchanged between the HiL simulator and the SUT, are required (i. e., signals that have been provided to the SUT and signals that are received by the SUT). As mentioned above, it is desirable that monitoring of the SUT's behavior is done in a non-intrusive way in order to preserve the property of observability.

On-line monitoring makes it possible to observe the behavior of the SUT during a simulation run. In the case where the HiL simulator allows user interaction, it is even possible that a human operator interactively alters parameters of a simulation during a simulation run. Logging the exchanged information between HiL simulator and SUT is required for postsimulation analysis purposes.

Signal Quality

The quality of signals (e. g., sampling rate, accuracy) provided by an HiL simulator must be in line with the SUT's interface specification. Furthermore, an HiL simulator

[1] This definition of deterministic behavior of an HiL simulator analogously follows the definition of TMR-deterministic behavior of a computational system as given in [KESHO07].

must be able to sample signals from the SUT at an update rate that is high enough to capture all significant state changes of the SUT's output interfaces.

Hardware used for the FSCs must be able to accurately and deterministically generate application dependent signals, for example:

- waveforms (variable reluctance sensor, simulated load/vibration),
- counters (pulse, pulse-width-modulated), and
- digital signals [NI03].

3.2.2 Economic Requirements

Apart from technical considerations regarding an HiL simulation setup, economic feasibility of a given approach is a crucial criteria, which can be measured in terms of

- cost for the required hardware and software and
- development time.

Cost for HiL Simulation

An HiL simulator requires specific hardware components that are capable of interfacing a certain SUT. Several commercial suppliers provide general purpose off-the-shelf as well as customer specific hardware boards that can be used for an HiL simulation (refer to section 3.3).

Besides the hardware cost, also the cost for software has to be taken into consideration. For the development and execution of an HiL simulation, several different software solutions are required. Software that is used for the development of the simulation model, software for the execution of this simulation model (e. g., operating system, middleware, low-level drivers), software for user interaction during a simulation run, and software for postsimulation analysis.

The cost of hardware and software for an HiL simulation can be quite high. In [Cra05], a reference to a statement of an Infineon Technologies marketing manager states that *the cost for each hardware-in-the-loop box or chassis for automotive systems comes at up to $50,000.* Nevertheless, there are also various examples of low cost solutions like for instance the real-time simulation platform VTB-RT [FBMD06].

Development Time for HiL Simulation

As mentioned in [BSS05], HiL simulation targets at automation and optimization of the test process, thus, allowing to preserve the quality of embedded software products in spite of decreasing development cycle times. Nevertheless, the development of an

HiL simulation typically requires a non-negligible amount of development time which contributes to the overall cost for setting up an HiL simulation.

Therefore, it is desirable for any HiL solution that the development of the simulation model, the execution of the simulation model, the user interaction, and the postsimulation analysis are supported by preexisting software components that can be configured for a specific HiL simulation. Furthermore, it is necessary that small changes in the SUT do not require a complete re-design of the HiL simulation. Moreover, flexibility and scalability of the development software is required to support stepwise implementation of an SUT, starting from a virtual system and targeting at a fully implemented system.

3.3 Existing Solutions

Within this section we present a selection of commercially used HiL simulation products. Therefore, we briefly investigate both, a mixture of multi-purpose solutions provided by well established vendors, and special purpose solutions from small companies. For each of the selected products we give a short profile of the vendor, the core features of the product(s), and the proposed solution for coupling of HiL simulator and SUT. The following information mainly originates from the respective web pages and was captured in October 2006 (without any claim regarding completeness or correctness of the given information).

3.3.1 DSP Builder (Altera)

Altera Corporation

Altera Corporation[2] was founded in 1983 and has its headquarter in San Jose, California. Altera is a large supplier of System-On-a-Programmable-Chip (SOPC) solutions with approximately 14.000 customers worldwide and an annual revenue of 1.12 billion dollars in 2005. The spectrum of Altera's target audience includes customers from the automotive, aerospace, broadcast (i. e., digital television), consumer electronics, and medical domains.

Features of DSP Builder

DSP Builder [DSP06] aims at shortening Digital Signal Processing (DSP) system design in Altera programmable logic devices (PLDs). DSP Builder integrates high-level algorithm and hardware description language development tools.

DSP Builder offers an HiL block that can be used within a MATLAB/Simulink model in order to co-simulate a Quartus II software design with a physical FPGA board (Quartus II is a development environment for SOPC design). Thereby, the contents and function of the FPGA is defined by creating and compiling a Quartus II

[2] http://www.altera.com

project. HiL simulation with the DSP Builder is used because of the ability to perform faster simulation and richer instrumentation when testing an DSP system design.

Coupling

The development workstation is used as the HiL simulator. The Altera FPGA includes the DSP system design and is regarded as the SUT. The interaction between the HiL simulator (i. e., the development computer) and the SUT (i. e., the FPGA) takes place via a Joint Test Action Group (JTAG) boundary scan interface. The widely accepted JTAG boundary scan standard is an IEEE standard (1149.1) that has been created for testing digital systems [SD02]. JTAG particularly addresses the testing of boards with high-density surface mount parts. It is also possible to implement built-in self-test functions based on JTAG ports.

With the JTAG boundary scan, test data is shifted along a serial scan path into so-called boundary cells (input- and output cells). Input data is provided to input cells via the Test Data Input (TDI) line, output data is received from output cells via the serial Test Data Output (TDO) line. The timing of this interaction (i. e., shifting data to / from boundary cells) is controlled by the Test ClocK (TCK) input of the test interface. This clock is independently from any system clock. Therefore, the HiL simulator has full control over the timing of the test-run.

The proposed architecture targets at testing single FPGA boards. Distributed SUTs are not supported by this approach.

3.3.2 LabVIEW (NI)

National Instruments (NI)

NI[3] was founded in 1976. Headquartered in Austin, Texas, NI operates worldwide and has currently more than 3,900 employees. NI offers computer-based solutions for embedded design, industrial control, test, and measurement that combine commercial off-the-shelf technologies with innovative software and hardware. NI targets at customers from automotive, aerospace, electronics, semiconductor, life sciences, and telecom industries.

Features of LabVIEW

The software product *LabVIEW* from NI provides a graphical development environment for signal acquisition, measurement analysis, and data presentation. Furthermore, the LabVIEW Simulation Interface Toolkit allows to interface with Simulink.

NI also provides embedded hardware for test, measurement, and control. This hardware is based on the PCI eXtensions for Instrumentation (PXI) specification, a standard for measurement and automation applications. PXI is maintained by

[3] http://www.ni.com

the PXI Systems Alliance (PXISA). Typically, a PXI system consists of a chassis, controller, and I/O modules.

Coupling

Different I/O solutions are available from NI, including analog and digital I/O, CAN, PWM, dynamic signals, motion control, image acquisition. Furthermore, it is possible to use FPGA modules and program them with LabVIEW for customized and fast I/O signals.

An HiL simulation system based on NI components involves either one or more ECUs (i. e., the SUT) that are connected to the HiL simulator. The HiL simulator is a single device that is tailored to a certain SUT by involving application specific processor boards and I/O boards. The interconnection of processor and I/O boards is based on the PXI standard combining the high-speed Peripheral Component Interconnect (PCI) bus with additional timing and synchronization features.

3.3.3 Pi Autosim (Pi Technology)

Pi Technology

Pi Technology[4] was founded in 1994 and recently employs 120 engineers. Pi Technology has its headquarter in Cambridge, England, and satellite offices in the US and in Germany. Pi Technology is focused on the development of electronic automotive systems and provides product development, engineering services, specialized test equipment, and consultancy for these systems. Customers of Pi Technology are vehicle manufacturers and Tier 1 Suppliers in the automotive domain.

Features of Pi Autosim

Pi Autosim is an HiL simulator for testing of automotive ECUs. Pi AutoSim consists of two components – the I/O system and the user interface. The I/O system simulates the environment of the automotive application (e. g., an engine, a braking system, or even a complete vehicle) and therefore consists of dedicated I/O cards and the respective software model. The user interface is executed on a Windows desktop PC and provides a graphical test environment for monitoring and user interaction. The Pi AutoSim user interface PC is connected to the I/O system via an Ethernet network connection.

Coupling

The hardware architecture of the Pi HiL simulator is that of a PC with an extended ISA bus, offering fifteen slots for I/O cards. Pi Technology provides different I/O cards, including I/O cards that are specifically tailored to automotive purposes.

[4] http://www.pitechnology.com

These I/O cards provide support for digital, PWM, and voltage signals as well as some more specialized items such as a programmable waveform generator and a fast event capture board. In addition, Pi Technology provides hardware support for specific interfaces that are found in automotive applications. A dedicated *Master Card* controls and synchronizes all I/O functions.

3.3.4 RTDS Simulator (RTDS Technologies)

RTDS Technologies

RTDS Technologies[5] was founded in 1994 and is located in Manitoba, Canada. The company is primarily engaged in manufacturing, marketing, servicing and continuing development of the Real Time Digital Simulator (RTDS) that is in use by the power industry. RTDS Technologies targets at customers in the power industry, including manufacturers of protective relays, controllers, and power electronic systems.

Features of RTDS Simulator

The product *RTDS Simulator* is used for HiL simulation of electromagnetic transient power systems such as protection equipment and control equipment. RTDS Simulator comprises computer hardware and software – both offered by RTDS Technologies.

The corresponding software, *RSCAD Software Suite*, provides the HMI of the RTDS Simulator and is designed to allow the user to prepare and run the simulation, and to analyze its output. Furthermore, several power and control system component models are available to create simulation cases.

Coupling

The RTDS Simulator builds upon a customizable parallel processing hardware architecture and is assembled in modular hardware units, called racks. The specific composition of an RTDS Simulator depends on the processing and I/O requirements of the intended application. RTDS Technologies offers a range of processing cards, e. g., the *Giga Processor Card* containing two IBM PowerPC 750GX RISC processors or the *Triple Processor Card* with three Analog Devices ADSP21062 (SHARC) Digital Signal Processors. The processing cards include digital and analog I/O. Furthermore, separate I/O cards are available that extend the number and capabilities of the existing I/O channels of the simulator. For instance, the *Digital Input and Output Card* offers 24 digital input and 24 digital output channels.

A common communications backplane links all cards that are installed in one rack. Communication between these cards can be done in parallel. For the communication between different racks, a dedicated Inter-Rack Communication card is offered. An Inter-Rack Communication card can connect a rack with up to six further racks by providing six 660 MBit/s full duplex serial communication channels.

[5] http://www.rtds.com

3.3.5 RT-LAB (Opal-RT)

Opal-RT Technologies

Opal-RT Technologies[6] was founded in 1997 with its headquarter in Ann Arbor, Michigan. Opal-RT Technologies provides software, hardware, and related services for real-time simulation, control system development and HiL testing. Opal-RT Technologies has expertise in distributed hard-real-time simulation, providing its solutions to manufacturers of automotive, aerospace, defense, mechatronic and electrical systems.

Features of RT-LAB

RT-LAB is a (tool supported) distributed real-time simulation platform based on a network of run-time targets. RT-LAB supports HiL simulation and rapid prototyping. Furthermore, RT-LAB allows to convert MATLAB/Simulink or SystemBuild[7] models to real-time simulations via Real-Time Workshop or Autocode.

Coupling

Opal-RT offers different specialized hardware solutions for interfacing between the real-time simulation and an SUT. The product portfolio includes hardware for testing powertrain applications (*RT-LAB TestDrive*), PC104 systems (*RT-LAB PC104 Rapid Prototyping Controller*), power electronics systems (*RT-LAB Electric Drive Simulator*), high power electric applications (*RT-LAB Electrical Engineering Simulator*), and avionics hardware components (*RT-LAB TestFlight*). Furthermore, the product *RT-LAB DriveLab* is offered for educational purposes.

RT-LAB supports a distributed target configuration, involving a network of RT-LAB PCs that execute a (distributed) simulation model. Each RT-LAB PC interfaces its I/O boards via the PCI bus. Multirate simulation is possible by using FPGA I/O boards that execute assigned parts of the simulation model. The communication between the different RT-LAB PCs can be established via FireWire or INFINIBAND. As presented in [BAD05], a FireWire (800Mbit/s single duplex) connection is used to connect up to 4 computers.

Firewire, also known as IEEE 1394 standard, allows fast data exchange between computers and peripheral devices. As of today, Firewire is mainly used for the transfer of digital images and videos and to connect to external storage devices. Firewire supports both, isochronous and asynchronous data transmission. Isochronous transfers offer guaranteed transmission intervals to send real-time data. The synchronization of devices in isochronous mode is handled by the root node (which is the device with the highest node ID). Up to 80% of bandwidth can be reserved to one or more isochronous channels. According to [BAD05], a firewire connection between RT-LAB PCs enables model time steps *as low as 30 to 60 μs*.

[6] http://www.opal-rt.com
[7] MatrixX SystemBuild is part of LabVIEW

For a larger number of interconnected computers or a higher amount of signal exchanges, the use of INFINIBAND (10 Gbit/s full duplex) is proposed. The INFINIBAND protocol is a popular public standard and supports copper cables and (optional) optical fiber cables. When using INFINIBAND, up to three computers can be connected by two adapters since each adapter has two full-duplex point-to-point links. All computers can communicate at the same time because of the point-to-point communication of INFINIBAND. In case the number of RT-LAB PCs exceeds three, INFINIBAND non-blocking switches must be used.

3.3.6 rtX Simulator (ADI)

Applied Dynamics International (ADI)

ADI[8] has been a supplier of HiL simulation solutions since 1957. ADI's product line includes Commercial-Off-The-Shelf (COTS) computer processors, industry standard interface devices, and software applications. Main focus of ADI is on designing and testing embedded control systems. Products of ADI have been in use by automotive, aerospace, and defense industries as well as by some more domains like chemical processing, robotics, power generation, and medical applications.

Features of rtX Simulator

The *ADI rtX simulator* is a standards-based open-architecture solution for HiL simulation, featuring Intel Pentium or AMD Opteron processors and supporting a range of PCI, CompactPCI, PXI and Industry Pack I/O solutions. The *RTS* computer platform, offered by ADI, provides multi-processor real-time simulation with an extensive array of I/O interfaces. RTS uses an operating system from ADI, which is called *RTexec*. In [ADI05], a distributed HiL simulation with an RTS VME-based real-time simulation computer and an rtX PC-based real-time computer is presented.

Coupling

As discussed in [ADI05], a distributed HiL simulator, assembled with ADI's products, includes master-slave clock synchronization. Master-slave clock synchronization is implemented by a clock master node sending an IRIG-B clock signal to its slave nodes. The clock master node further controls the start of a simulation by sending a *triggered start* signal to its slave nodes. The communication between the simulator nodes takes either place via *emulated electronic interfaces*, i. e., by exchange of data visible at the interfaces of the SUT, or via *virtual interfaces*, i. e., by exchange of data that is part of the simulation model and therefore not visible at the SUT's interfaces. Ethernet or SCRAMNet is used to implement the virtual interfaces.

[8] http://www.adi.com

3.3.7 Simulator Mid-/Full-Size (dSpace)

dSPACE

dSPACE[9] was founded in 1988. Its headquarter is located in Paderborn, Germany. dSPACE offers engineering tools for developing and testing mechatronics control systems. dSPACE has a significant market share regarding HiL simulation systems with 12.000 systems in use worldwide. The main application field of dSPACE is within automotive industry. However, products from dSPACE are also employed in aerospace industry, education, and research.

Features of dSPACE Simulators

dSPACE offers two simulator products, *Simulator Mid-Size* and *Simulator Full-Size*. Both can be equipped with a range of modular I/O boards and processor boards, tailoring them to a certain HiL simulation system. Faults and errors in a dSPACE HiL system, can be simulated within the simulation model and/or with the help of a dedicated failure insertion unit that inserts electrical failures on ECU inputs and outputs.

The code for dSPACE simulator products is automatically generated from MATLAB/Simulink. Thereby, the dSPACE's *Real-Time Interface* integrates MATLAB/Simulink and offers additional Simulink blocks, connecting simulation models to I/O hardware. Furthermore, dSPACE offers real-time models that can be used for simulation on dSPACE simulators (e. g., simulation of powertrain or vehicle dynamics) and an HMI software solution (*ControlDesk*) to visualize relevant information and interactively change parameters during an HiL simulation.

Coupling

From an architectural point of view, an HiL simulation setup with dSpace components involves one HiL simulator and one or more ECUs that are connected to the HiL simulator (i. e., the SUT). Physically, the HiL simulator is a single device which can be tailored to the SUT's requirements by adding processors and I/O cards. Internally, the proprietary PHS bus (or PHS++ in newer devices) is used to interconnect different dSpace cards.

3.3.8 Tanto2 Test (Hitex)

Hitex

Hitex[10] was founded in 1976 in Karlsruhe, Germany, and is currently present in Europe, the US, and Asia. Since 2003, Hitex belongs to Infineon Technologies AG. Hitex offers analysis and test software for embedded systems design, including automated

[9] http://www.dspace.com
[10] http://www.hitex.de

tests, in-circuit-emulators, simulators, debuggers, evaluation units, and analysis tools for automotive applications. Products of Hitex are used by customers from automotive, aerospace, consumer electronics, and semiconductor industries.

Features of Tanto2 Test

Hitex offers a product called *Tessy* which can be used to automate unit tests of embedded software. The purpose of Tessy is to provide input parameters, execute tests and evaluate results, as well as to manage test data, document test results, and state the test coverage. *Tanto2 Test*, an extension module to Tessy, is offered for HiL testing. Tanto2 Test generates and captures dynamic signals and signal sequences with sampling frequencies of up to 200 MHz.

Coupling

Tanto2 Test provides the physical interface to an SUT. A standard PC, running Tessy, generates and captures target hardware signals. A Tanto2 Test hardware board is connected to the PC via an Ethernet or an USB 2.0 interface. Tanto2 Test targets at testing single ECUs.

3.3.9 xPC Target (MathWorks)

The MathWorks

The MathWorks[11] was founded in 1984. With its headquarter in Natick, Massachusetts, The MathWorks is represented worldwide and currently employs more than 1,400 people. The MathWorks provides software for technical computing and model-based design, supporting customers from diverse domains, including aerospace and defense, automotive, biotech, pharmaceutical and medical, communications, computers and office equipment, electronics, financial services, industrial automation and machinery, and semiconductors.

Features of xPC Target

In order to perform real-time HiL simulation, the MathWorks offers *xPC Target* which is based on MATLAB/Simulink. With xPC Target it is possible to connect a simulation model to physical systems and to execute it in real time on PC-compatible hardware. The simulation model can be developed with MATLAB/Simulink and with Stateflow. Stateflow is an interactive design and simulation tool for event-driven systems that is integrated into MATLAB/Simulink. xPC offers I/O interface blocks that can be added to a MATLAB/Simulink or Stateflow model. The MathWorks additionally offer code generators (i. e., Real-Time Workshop and Stateflow Coder) which can automatically generate code from a MATLAB/Simulink and Stateflow model.

[11] http://www.mathworks.com

When using xPC Target, the simulation model is developed on a host PC running a Microsoft Windows operating system. The execution of this simulation model is performed by a target PC that can be any PC with Intel or AMD 32-bit processors, e. g., a desktop computer, an industrial computer such as xPC TargetBox, PC/104, PC/104+, CompactPCI, all-in-one embedded PC, or any other PC-compatible form factor.

Coupling

The host PC can be regarded as the development and HMI computer. The coupling between host PC and target PC can either be a serial RS232 connection or a TCP/IP network connection. The coupling between the target PC and the SUT is established via I/O boards that are inserted into the target PC. xPC Target supports ISA, PCI, PC/104, PC/104+, and CompactPCI I/O hardware boards. Thus, the HiL simulator is a single device, consisting of the target PC and several I/O boards.

3.3.10 Comparing the Existing Solutions

The HiL simulation systems presented in this section range from simple simulators, targeting at testing single ECUs, to complex simulators, capable of testing large distributed real-time systems. The HiL simulators offered from Altera and Hitex are examples for comparably simple solutions, where a single hardware target is directly connected to a standard PC. Several vendors offer solutions for more complex HiL simulators. Regarding such complex HiL simulators, we can basically distinguish between two styles:

- The first style is based on a modular, component based, monolithic HiL simulator, where a single device is configured to offer all required interfaces for a particular SUT. The proposed HiL simulators from dSpace, The MathWorks, NI, and Pi Technology follow such approach.
- The second style is based on a distributed HiL simulator, executing a distributed simulation model. Distributed HiL simulators are offered by ADI, Opal-RT, and RTDS.

In table 3.1, the discussed HiL simulation products are compared with respect to size of the SUT, structure of the HiL simulator, and employed approach for the interconnection of FSCs (which are typically specialized I/O boards).

Throughout this thesis we particularly focus on a distributed HiL simulation that involves a collection of tightly synchronized simulator components (i. e., FSCs and BSCs). The interplay between these components depends on timing properties (latency, jitter) of the selected communication mechanism.

In commercially available products, a common approach to interconnect simulator components is to use a cascaded master slave topology. The master synchronizes

Product	Vendor	Supported SUT	Structure of Simulator	FSC Interconnection
DSP Builder	Altera Corporation	Single FPGA board	Monolithic (simple)	JTAG
LabVIEW	National Instruments	Distributed SUT	Monolithic (modular)	PXI
Pi Autosim	Pi Technology	Single ECU	Monolithic (modular)	ISA
RTDS Simulator	RTDS Technologies	Single Power System Equipment	Distributed	Interrack communication backplane
RT-LAB	Opal-RT Technologies	Distributed SUT	Distributed	FireWire, INFINIBAND
rtX Simulator	Applied Dynamics International	Distributed SUT	Distributed	Ethernet, SCRAMnet
Simulator Mid-/Full-Size	dSpace	Distributed SUT	Monolithic (modular)	PHS, PHS++
Tanto2 Test	Hitex	Single ECU	Monolithic (simple)	Ethernet, USB2.0
xPC Target	The MathWorks	Single HW Target	Monolithic (modular)	PCI, PC/104, PC/104+, ISA, CompactPCI

Table 3.1: Comparison of HiL simulation products

its slaves by providing a clock signal (e. g., IRIG-B clock signal in ADI rtX Simulator). Each slave is connected to the master via a separate line (e. g., Ethernet or SCRAMnet with ADI rtX Simulator, FireWire or INFINIBAND with Opal-RT RT-LAB).

Regarding scalability, a single communication channel, shared by all simulation components, is advantageous. However, bus arbitration of a single channel might introduce unacceptable jitter during a simulation run. The solution we propose in chapter 4 is based on a time-triggered network which offers predictable communication with small latencies and virtually no jitter.

3.4 Chapter Summary

An HiL simulation system consists of a *System-Under-Test (SUT)* and an *HiL simulator*. The SUT is typically (part of) a real-time computer system, i. e., a single ECU or a distributed system. An HiL simulator consists of an *environment simulator* and (optionally) a *cluster simulator*. An environment simulator interacts with the SUT across the COI, thereby simulating the physical environment of the SUT. A cluster simulator interacts with the SUT across the LLI, thereby simulating the behavior of missing nodes of the SUT. We can distinguish between *open-loop HiL simulation*, in which the HiL simulator provides static simulation data to the SUT, and *closed-loop HiL simulation*, in which the HiL simulator dynamically produces simulation data based on feedback from the SUT.

A distributed HiL simulator consists of *BSCs*, which are independent of low level details regarding the physical interconnection between HiL simulator and SUT, and *FSCs* which are in charge of controlling the interaction between HiL simulator and SUT. In a *multirate HiL simulation*, these components use different rates for the execution of a distributed simulation model.

Before starting to realize an HiL simulation system, several questions must be answered, including the *intended use*, *level of detail*, and *fidelity* of an HiL simulation, as well as the *availability of data* for the simulation. After that, the technical and economic requirements have to be captured in detail. Four basic technical requirements are: the ability of *real-time execution* of the simulation model, the *reproducibility and observability* of simulation results, the availability of mechanisms for *monitoring and data logging*, and the provision of sufficiently *high signal quality*. From an economic point of view, *low cost* of HiL simulator components and *short development time* of an HiL simulation system are required.

Existing HiL simulator systems range from simple HiL simulators, aiming at testing single ECUs, to complex solutions, capable of testing distributed systems. Complex simulators are either based on a modular, component based, monolithic HiL simulator or on a distributed HiL simulator.

Chapter 4

Interface Design for HiL Simulation

In this chapter, we present an HiL simulation system framework, focusing on deterministic interaction between the HiL simulator and the SUT. We start with the rationale of this framework and investigate the proposed solution in detail. After that, we outline the concept of a Smart Virtual Transducer (SVT) that is employed to emulate an arbitrary transducer interface during an HiL simulation run. This chapter is concluded by revising the rationale with respect to the proposed approach.

4.1 Rationale

We propose a scalable, component-based, deterministic HiL simulation system framework based on the TTA. The HiL simulation system framework particularly targets at:

- the ability to provide deterministic interaction between the HiL simulator and the SUT,
- support for a distributed HiL simulator, consisting of several BSCs and FSCs,
- support of arbitrary transducer interfaces, and
- mechanisms for monitoring, on-line configuration, and data logging.

4.1.1 Deterministic Interaction

As discussed in section 3.2.1, an HiL simulator is said to behave deterministically, if a defined initial state $q_0(t_0)$ of the HiL simulator at $t_0(now)$ and a sequence of input symbols $a_i(t_i)$ at sparse real-time instants t_i will always produce the same sequence of output symbols $b_j(t_j)$ at the same future sparse real-time instants t_j [KESHO07]. Hence, the behavior of the HiL simulator is causally determined by past behavior of HiL simulator and SUT.

The behavior of the HiL simulator (resp. the SUT) is defined by traces of its *declared state*. The declared state is a *declared data structure that can be accessed via an interface and that records all the stored state that is relevant to the future essential behavior of the system* [JKK+02] (refer to section 2.2). Deterministic interaction must not to be confused with ideal resemblance of an HiL simulation with the real environment of the SUT, i. e., deterministic interaction between HiL simulator and SUT does not necessarily mean high accuracy of the simulation.

Deterministic interaction between HiL simulator and SUT is a prerequisite for reproducible simulation runs, thereby relating to the functional (i. e., message value or signal level) as well as to the temporal domain (i. e., instant of interaction). In the course of the thesis we are concerned with temporally deterministic interaction (i. e., no probabilistic element with respect to the instants of interaction).

If an HiL simulator is designed for deterministic interaction with a particular SUT, it can be guaranteed that a set of inputs, provided by the SUT to the HiL simulator at defined instants, always causes the same set of outputs from the HiL simulator to the SUT at the same instants.

Deterministic interaction requires that messages exchanged between HiL simulator and SUT are discrete in both, time and value domains. Furthermore, HIL and SUT must be synchronized in order that the resulting state is unaffected by the instant when a message is exchanged as long as this instant lies within a defined update interval. This property can be achieved by implementing clock synchronization and establishing a *sparse time base*. Thereby, the update interval is represented by the *active interval*.

Additionally, the HiL simulator must provide timely execution of the simulation model. If this cannot be guaranteed *a priori* (e. g., due to unpredictable behavior of the employed HiL simulator hardware, operating system, ...), error detection must be put in place to detect late delivery of simulation values. Based on diagnostic information, the validity of a simulation run can then be checked.

4.1.2 Distributed HiL Simulator

An HiL simulator can either be built as a monolithic or as a distributed system. In our approach a distributed HiL simulator is employed, i. e., the HiL simulator is based on a distributed system consisting of a number of interacting nodes. As a special case, a distributed HiL simulator can be reduced to a monolithic HiL simulator by being realized on just one single node. An HiL simulation system framework that supports the construction of a distributed HiL simulator allows for: independent development of components, scalability, flexibility, complexity management, and support for multirate simulation.

Independent Development of Components

As depicted in figure 4.1, subsystems of a distributed real-time system can be individually tested by an assigned HiL simulator. After assembling the distributed

real-time system through the integration of subsystems, it is desirable that the existing HiL simulators that have been in use to test the subsystems, can be reused to test the overall system. Two additional components marked with X in figure 4.1 point out that the integrated HiL simulator may extend the functionality of its subparts by additional functions required for system-level tests. A distributed HiL simulator eases reuse of its components within a larger HiL simulator.

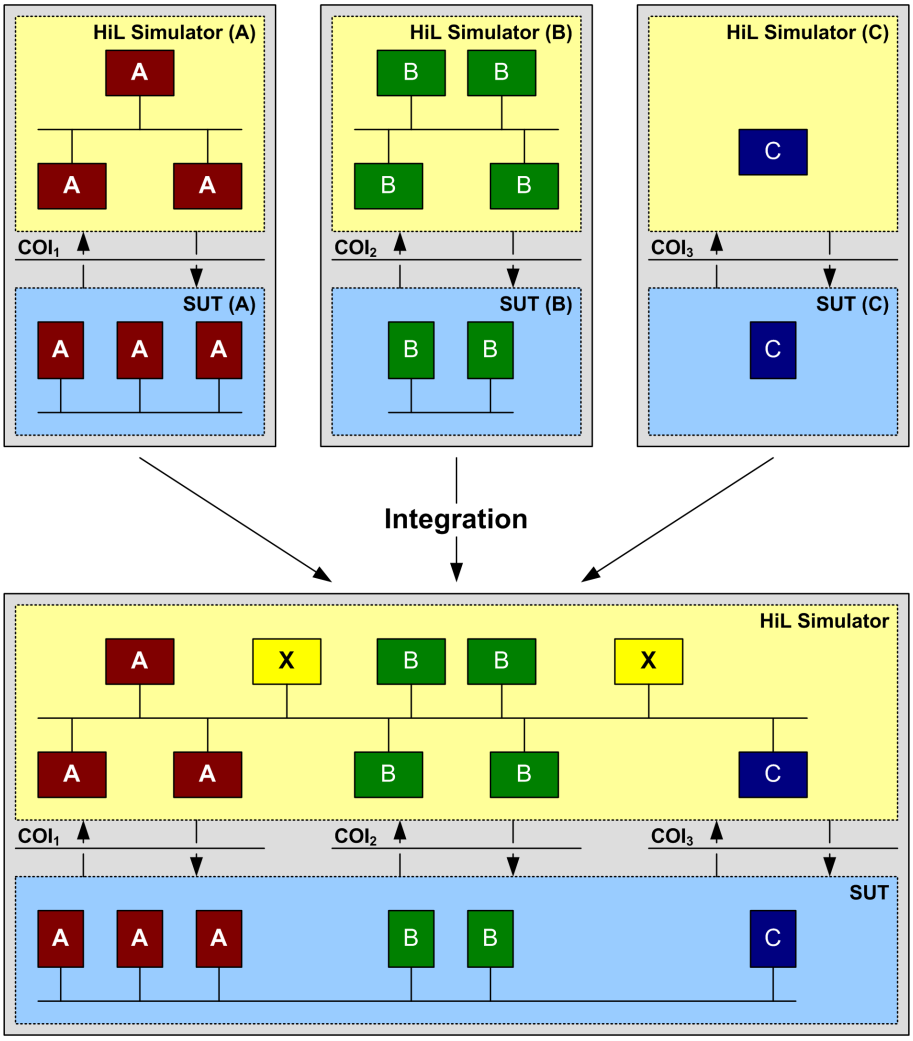

Figure 4.1: Integration of different HiL simulators

Scalability

Typically, the development of an SUT starts with a small subset of basic functions. The SUT is extended as soon as these basic functions are available. Testing a particular SUT with an HiL simulator only requires to cover the available parts of the SUT prototype. Hence, the possibility of a stepwise development of the HiL simulator is beneficial, providing the most important features at an early development stage and allowing enhancements at further development stages. In order to achieve

a scalable architecture, a distributed HiL simulator is required that can be extended by components adding processing power or additional transducer interfaces.

Flexibility

The nodes of a distributed SUT are not necessarily located in close proximity. A part of these nodes could for instance be mounted on a physical test bench, located in a different laboratory than the rest of the SUT. A distributed HiL simulator allows to design an application specific number of FSCs which can be mounted in close proximity to the nodes of the SUT.

Complexity Management

The cognitive complexity of a large system with many interactions within this system can only be reduced by the introduction of structure and hierarchies [Sim96]. Thus, a set of well-specified components that interact across their linking interfaces contributes to reducing the complexity of the HiL simulation system. A systematic separation of BSCs and FSCs helps focusing either on the system level view of the environmental simulation, or on specific physical details (e. g., accuracy of a certain PWM signal).

Support for Multirate HiL Simulation

As discussed in section 3.1, multirate HiL simulation involves a number of simulation components, executing parts of a distributed simulation model at different scan rates.

A distributed HiL simulator that implements a multirate HiL simulation is able to combine open-loop and closed-loop HiL simulation. An example of such a combination would be an HiL simulator that involves fast FSCs and comparably slow BSCs. Each node is equipped with its own local processing unit that operates at a defined rate. The simulation model of this HiL simulator is executed by the BSCs in a closed loop, i. e., the feedback of the SUT is captured and processed by the BSCs in order to calculate new simulation values. Each FSC implements an open-loop simulation, i. e., an FSC provides a set of simulation values and/or records the feedback of the SUT.

Due to the fast rate of an FSC, the feedback of the SUT is recorded and a fused value (e. g., the average) is sent to the BSCs for further processing. An FSC receives the simulation data from the BSCs (at a slower rate) and interpolates the simulation data. Hence, the feedback of the SUT does not directly influence the provision of data by the FSC.

4.1.3 Arbitrary Transducer Interfaces

An HiL simulator must support the application of a wide range of transducer interfaces. As depicted in figure 4.2, the physical coupling between an HiL simulator and

the SUT is defined by the COI. The COI consists of a set of specific interfaces that correspond to each transducer attached to the SUT.

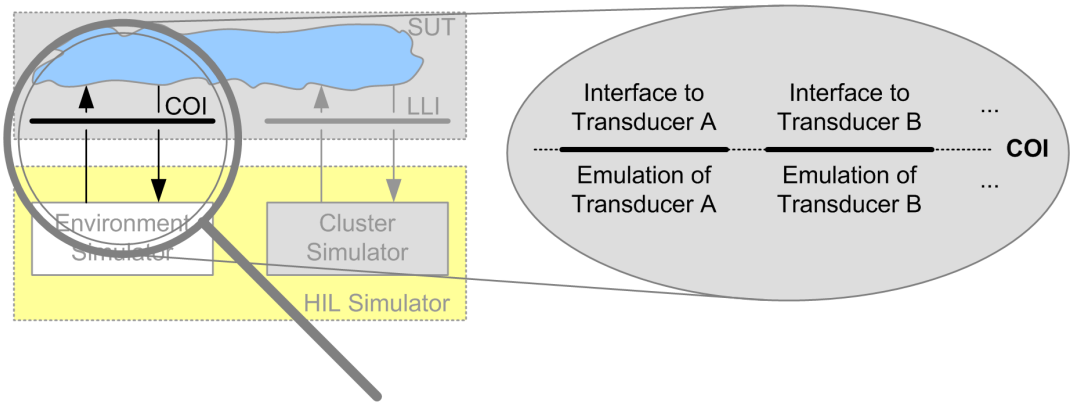

Figure 4.2: Physical coupling across transducer interface

Examples for the interaction across a transducer specific interface are: capturing/setting an analog signal, measuring the amount of time between two consecutive interrupts that are raised by an external device, or reading/writing a message in accordance to a well defined digital interfacing scheme. The latter applies in case the SUT interacts with a network of transducers (fieldbus) and the SUT indirectly accesses the relevant information from a particular transducer by the exchange of (fieldbus-)messages.

A systematic approach for the support of a large range of transducer interfaces reveals the following requirements:

- The modular construction of an HiL simulator with configurable, re-usable FSCs shall be supported.
- An open solution for the emulation of transducer specific interfaces is required.
- A well defined LLI for the interconnection of HiL simulator components shall be available.

Configurable, Re-usable FSCs

An FSC establishes the physical link between the HiL simulator and the SUT. Hence, it provides the physical connection (i. e., cable(s)) as well as the emulation of the transducer behavior in the functional and temporal domains. Each transducer emulation must be tailored to the properties of the SUT's COI.

The effort to develop a transducer emulation for each single transducer within an HiL simulation can be reduced by the provision of reusable (generic) FSCs, each supporting a class of transducers. From an architectural viewpoint, the construction of an HiL simulator with modular, loosely coupled components, which can be individually adjusted to a certain SUT, supports reuse.

Openness

The construction of an HiL simulator involves the development of multiple FSCs. As mentioned above, the reuse of existing configurable components helps to decrease the effort of developing these FSCs. Nevertheless, the great range of different existing transducer interfaces as well as the continuing development of new transducers (with new interfaces) calls for a solution supporting the development of new transducer emulations. It follows that FSCs must be open to new implementations of transducer emulations.

Interconnection of HiL Simulator Components

The integration of HiL simulator components (i. e., FSCs and BSCs) shall be guided by a standardized LLI specification. Reuse of FSCs from different HiL simulators shall be made possible by a consistent interaction scheme for the communication between FSCs (and BSCs).

An important property for the LLI specification is that of composability. It shall be possible to guarantee that the integration of different simulator components does not invalidate any properties of these simulator components. Furthermore, it shall be possible to integrate existing solutions (e. g., a commercially available FPGA I/O board). With a standardized LLI specification, two possible solutions exist for such integration:

- Either the (legacy) device that is used during the HiL simulation can be extended to be in line with the LLI specification, or
- a connection system is developed eliminating property mismatches between the device and the rest of the HiL simulator (refer to figure 4.3).

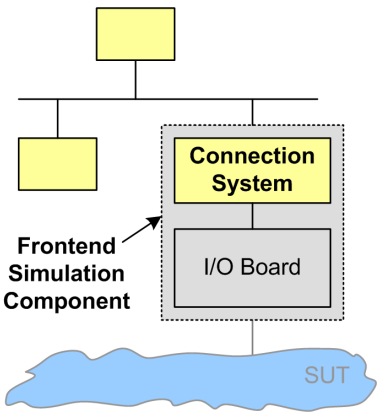

Figure 4.3: Connection of a (legacy) I/O board

4.1.4 Monitoring

Typically, the user of an HiL simulation is concerned about input data from the HiL simulator and output data from the SUT. Hence, a mechanism is required that provides this data, or parts thereof, to the user. The level of detail of the relevant data depends on the actual HiL simulation. A user may not be interested in all data produced during an HiL simulation run. For instance, it may be sufficient to access every tenth measurement of a temperature value, or to capture only a fraction of all available variables of a simulation model.

Monitoring of simulation data requires that this data is represented through variables of the simulation model. Thus, monitoring of an HiL simulation is equivalent to monitoring relevant variables of the HiL simulation model.

A mechanism for monitoring an HiL simulation shall enable observability (refer to section 2.3). In order to provide the property of observability, a communication infrastructure that links components of a distributed HiL simulator and supports monitoring without probe effect is desirable.

Furthermore, support to access variables not explicitly visible on the shared communication medium (i. e., variables that are not exchanged between components of the HiL simulator) shall be possible in order to monitor and configure component internal parameters.

Monitoring of an HiL simulation is concerned with two aspects which are elaborated in the following:

- First, it shall be possible for the user to access relevant data during a simulation run and to configure an HiL simulation run based on this data.
- Second, systematic capturing and recording of simulation data for the purpose of post simulation analysis shall be supported.

Run-time Configuration of HiL Simulation

As depicted in figure 4.4, run-time configuration of an HiL simulation requires that the HiL simulator captures and provides relevant simulation variables on-line, i. e., in real time. These variables are visualized on an HMI computer. The user reads these variables and modifies certain configuration parameters of the simulation. After that, the HiL simulator receives the altered parameters through the HMI computer.

Capturing relevant simulation variables, providing these variables to the user, and on-line configuring the simulation model by the user, happens in real time. Typically, the HMI computer is implemented as a soft real-time system.

Capturing of Data for Post Simulation Analysis

In order to enable post simulation analysis, it shall be possible to capture and store a set of relevant state observations during an HiL simulation run. These state observations include input variables, provided to the SUT by the HiL simulator, and

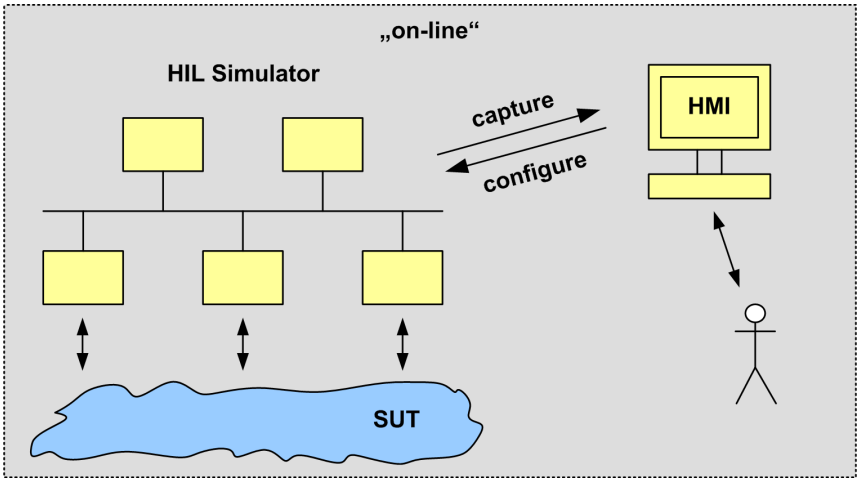

Figure 4.4: Configuration

output variables from the SUT, that are captured by the HiL simulator. After an HiL simulation run is finished, stored data is used for off-line post simulation analysis (refer to figure 4.5).

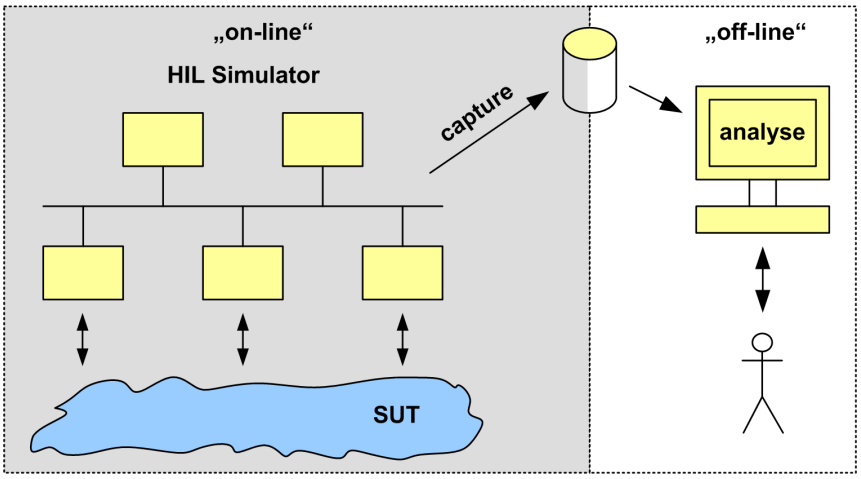

Figure 4.5: Post simulation analysis

Capturing variables during an HiL simulation run for the purpose of post simulation analysis requires a mechanism that deterministically captures and stores all state observations of interest. The set of relevant state observations is defined *a priori*, i. e., before the start of an HiL simulation run.

4.2 Architecture

Within this section we outline an HiL simulator based on the TTA that follows the rationale given in section 4.1. Starting from a top-level view, we elaborate the purpose of the involved components in detail.

4.2.1 Structure of HiL Simulator

The proposed solution is based on the TTA consisting of nodes that interact via a shared communication medium. We have chosen the time-triggered fieldbus protocol TTP/A that implements this interaction via a single bus line. TTP/A offers real-time capability, means for on-line diagnostics, extensibility, as well as low complexity and cost (refer to section 2.4).

Due to the selection of a time-triggered communication and execution scheme, all events within the HiL simulator (i. e., message transmissions, task execution) are triggered according to a global sparse time base. The time-triggered communication and execution scheme is a fundamental pre-requisite in order to establish determinism of the HiL simulator with respect to the execution of the simulation model and the interaction with the SUT.

TTP/A is a master slave protocol. Hence, a master node as well as a set of slave nodes is required to execute the simulation model and to physically interact with the SUT. We distinguish between nodes that physically implement the COI (e. g., via a 4-15mA interface, a fieldbus, direct I/O, or a real transducer) and nodes that execute part of a distributed simulation model but that do not directly interact with the SUT.

Following this separation, each HiL simulator involves three different elements:

Master: A master node is required in a TTP/A network in order to initiate the transmission of messages as well as the execution of time-triggered tasks at each node. Furthermore, the master node synchronizes its time-base to the time-base of the SUT.

Backend Simulation Component (BSC): A BSC is used to execute part of a distributed simulation model. A BSC has a digital interface to the TTP/A network and can (optionally) have a second digital gateway interface in order to connect to an external system and to extend the HiL simulator by a single (powerful) simulation computer or even a cluster of simulation computers performing computation-intensive calculations. Furthermore, the gateway interface supports adding an HMI computer for monitoring or run-time configuration of the HiL simulator.

Frontend Simulation Component (FSC): An FSC controls the physical interconnection between the HiL simulator and the SUT. Each FSC consists of a digital interface to the TTP/A network and a second interface that resembles the COI of the SUT. This second interface is either a digital fieldbus interface, an arbitrary transducer interface (e. g., direct I/O), or a physical transducer (e. g., a potentiometer that connects to a servo).

Figure 4.6 depicts the basic elements of the HiL simulator. We distinguish between a BSC without a second interface which we call *basic BSC*, and a BSC with a gateway interface which we call *gateway BSC*. Furthermore, we differentiate between an FSC used to connect to a fieldbus (*fieldbus FSC*), an FSC directly interfacing the

SUT through a transducer specific interface (*virtual transducer FSC*), and an FSC controlling a real transducer which is used to interact with a transducer of the SUT (*real transducer FSC*).

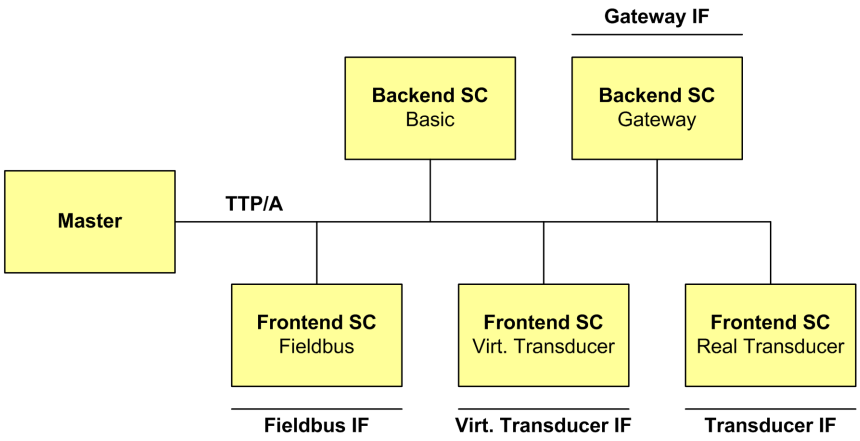

Figure 4.6: Distributed HiL simulator

An HiL simulator can be assembled as a multicluster system consisting of several TTP/A networks that interact via TTP/A gateways. Although the conceptual framework presented within this thesis does not restrict the setup of such a multicluster HiL system, we do not explicitly focus on multicluster HiL simulators within the thesis.

4.2.2 HiL Simulator Node Interaction

As mentioned above, the interaction of the HiL simulator nodes is based on TTP/A. TTP/A allows for a total number of 255 nodes within the same cluster. The TTP/A master node maintains the cluster wide time-base and periodically synchronizes the time-bases of its slaves (i. e., the BSCs and FSCs). The master node synchronizes its own time base to the time base of the SUT. If such synchronization between SUT and HiL simulator is not possible, deterministic interaction between SUT and HiL simulator as outlined in section 4.1 cannot be guaranteed because an instant, significant within both systems, cannot be precisely defined.

As outlined in section 2.4, the nodes of a TTP/A cluster share the bus according to a TDMA scheme. The master node periodically initiates one of 8 pre-configured rounds. Each round specifies an interaction pattern associating a particular time-slot to a node allowing this node to broadcast a message. TTP/A supports two round types: multipartner rounds and master-slave rounds. This concept is very beneficial for the construction of an HiL simulator because it supports both, periodic real-time communication for a given simulation run as well as sporadic communication with a particular node of the HiL simulator in order to enable monitoring and run-time configuration of this simulation run.

The employment of a time-triggered protocol within the HiL simulator enables predictability regarding communication between simulator components. Moreover,

due to the concept of a temporal firewall (refer to section 2.2) cognitive complexity is reduced because of temporal decoupling of simulation model execution at the BSCs and interaction with the SUT at the FSCs. Hence, it is not necessary to restrain execution of the simulation model at the BSCs to produce a certain simulation result at a precise instant. It is sufficient to ensure that the execution of a simulation step finishes within a given interval (i. e., the interval from start of execution of a single simulation step at a BSC until the instant of transmitting the simulation result to the FSC(s)) which allows for *temporal laxity* of the HiL simulator. Additionally, timing violations of a simulation step can be deterministically diagnosed for the purpose of repeating the simulation.

4.2.3 Backend Simulation Component (BSC)

We distinguish between two different types of BSCs:

- Simple BSC
- Gateway BSC

For the physical realization of a simple BSC, any node can be used that comprises a TTP/A network interface. For instance, a TTP/A node as depicted in figure 2.8 can be deployed for the implementation of a simple BSC, consisting of a small embedded micro controller (without a physical transducer element).

A gateway BSC has two interfaces, a TTP/A network interface and an additional gateway interface. The purpose of a gateway BSC is twofold: On the one hand, it enables the extension of limited computational performance of its embedded micro controller through a high performance simulation computer (figure 4.7 a)) or even a cluster of simulation computers (figure 4.7 b)). On the other hand, a gateway BSC provides means to establish a connection between the HiL simulator and an HMI computer (figure 4.7 c)). Hence, a gateway BSC allows to forward output of a simulation run to a database on an HMI computer and to configure a simulation at runtime. We have shown the realization of a gateway BSC in [Sch03]. In this work, the gateway BSC was physically implemented on a TTP/A master node communicating with a Linux-based PC across an RS232 serial line.

4.2.4 Frontend Simulation Component (FSC)

An FSC is used to connect the HiL simulator to the SUT. Hence, each FSC comprises a TTP/A network interface and an interface that emulates the COI of the SUT. Regarding the latter we distinguish between the following three types of FSCs:

- Fieldbus FSC
- Virtual Transducer FSC
- Real Transducer FSC

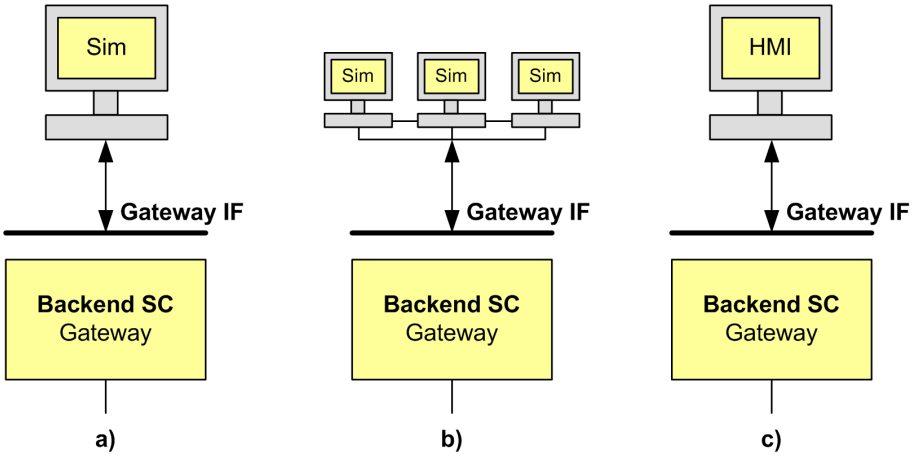

Figure 4.7: Types of gateway BSCs

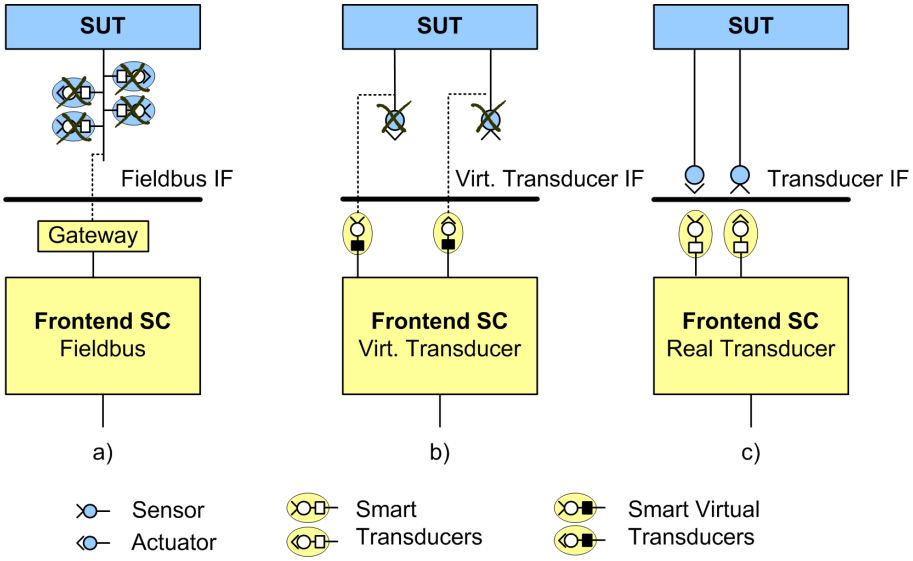

Figure 4.8: Types of FSCs

Figure 4.8 depicts the three types of FSCs. In figure 4.8 a), the SUT accesses its transducers via a digital transducer network interface (fieldbus). Thereby, a fieldbus FSC implements a gateway emulating the smart transducers of the SUT not present in the HiL simulation system. If the interface between SUT and transducers is a predictable digital real-time network, the reproducibility of simulation runs can be guaranteed on the basis of a timing analysis. Thus, this approach is preferable over the variant shown in figure 4.8 b), however it requires that the SUT has an appropriate transducer network interface, thus is less flexible than b).

In figure 4.8 b), we assume that the interfaces to the physical sensors or actuators are provided by a virtual transducer FSC. For the realization of a virtual transducer FSC we employ SVTs as is explained in section 4.3. The X over the transducers of the SUT indicates, that these are not present in the HiL simulation system. In many cases, the virtual transducer FSC has to generate or consume analog signals (in the

value and/or the time domain), which affects the reproducibility of a test run. On the other hand, this approach affords minimal intervention with the SUT and thus avoids probe effects at the SUT.

There are many different examples of transducer-specific interfacing schemes like for instance the range of an analog signal representing the measurement of an infrared sensor, the response behavior of an ultrasonic sensor, or a PWM signal for an electrical motor. For the coupling via transducer-specific interfaces, different approaches can be found within the literature, ranging from simple solutions to generic reconfigurable devices like for instance the PXI-7831R FPGA I/O board from National Instruments [NI03].

Approaches a) and b) are usually chosen, when the physical transducer is not available or a certain test scenario cannot be established by interfacing the transducer. For example, if a fault injection campaign involves transducers to exhibit a particular erroneous behavior, the use of real transducers is problematic.

Figure 4.8 c) depicts a configuration, where the SUT physically keeps its transducers. In the case of a sensor, a real transducer FSC drives an actuator that physically interacts with the sensor of the SUT. In the case of an actuator, a real transducer FSC measures the actions of the actuator via a sensor that is connected to the HiL simulator. The benefit of interfacing the SUT via a physical transducer is that a simulation model of the physical transducer is not necessary. Thus, the coupling via the physical transducer is the preferred approach if it is infeasible (technically or economically) to set up a sufficiently accurate simulation model of the transducer. In [Gom01], an HiL simulator for an aerospace application (autopilot) is described that physically interfaces the SUT via an elevator servo. Instead of modeling the internal physical behavior of the elevator servo, the deflection of the physical servo is read from a feedback potentiometer.

4.3 Smart Virtual Transducer (SVT)

As mentioned in section 4.2, an SVT is used to realize a virtual transducer FSC. The concept of an SVT has been published in [SEW06]. In this section we summarize the structure, the interfaces, and the classification of an SVT. Furthermore, we show a prototypical realization of an SVT and outline the benefits of the SVT concept.

4.3.1 Structure

The concept of an SVT is closely related to the concept of an ST as described in section 2.4. In contrast to an ST, an SVT does not contain a physical sensor or actuator element. Instead, an SVT is used to emulate a sensor or actuator element. Thus, an SVT consists of a processing unit, a communication interface, and a transducer-specific interface (refer to figure 4.9).

The purpose of an SVT is either to emulate the behavior of a physical sensor or to emulate the behavior of a physical actuator. Thus, an SVT implements either a

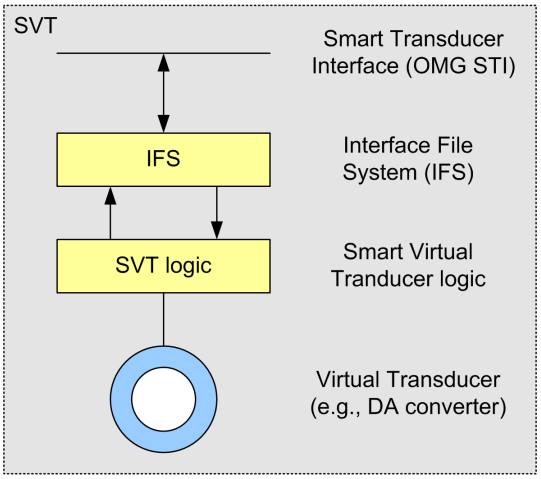

Figure 4.9: Elements of an SVT

virtual sensor or a *virtual actuator*. An SVT emulating a sensor is provided with simulation data by BSCs (refer to figure 4.6), an SVT emulating an actuator provides actuator data to BSCs.

An SVT consists of the following to parts:

SVT logic with communication interface: The digital communication interface is used for the exchange of messages between an SVT and the rest of the HiL simulation system according to the time-triggered interaction scheme of the OMG STI [OMG03].

Virtual transducer with VT interface: The virtual transducer interface of the virtual transducer resembles the interface between the SUT and a particular transducer. Particularly in the case of a time-dependent sensor (e. g., ultrasonic sensor), the timeliness of the virtual transducer response is important.

4.3.2 Interfaces

Interaction between SVT and SUT takes place via the virtual transducer interface. Thereby, a request from the target system to perform a sensor reading of a virtual sensor or to set a virtual actuator are not under control of the SVT. Such a request can, e. g., be to read a D/A value (virtual IR sensor), to measure the delay of a response (virtual ultrasonic sensor), or to set a PWM signal (virtual PWM actuator).

Regarding the occurrence of requests from the target system, we distinguish between two different cases:

Controlled access: The HiL simulation system has *a priori* knowledge about the instants when the SUT reads its sensors or updates its actuators. In order to achieve this, the design of the SUT has to be known to the extend of the timing of all possible task activations of tasks that access the transducers. A time-triggered communication and process model of the SUT is a prerequisite

for controlled access. With controlled access, the SVTs of the HiL simulator provide new sensor values of the virtual sensors at *a priori* defined instants, i. e., immediately before the SUT reads its sensors. Furthermore, the SVTs of the HiL simulator know *a priori* the instants of reading actuator control values from the SUT.

Arbitrary access: The SUT is treated as a black box and it is assumed that the SUT can access a virtual transducer at any point in time. Therefore, the SVTs have to provide valid sensor values at any instant and, respectively, have to log new actuator settings instantly. With arbitrary access, the BSCs of the HiL simulator recurrently send updated simulation values to the FSCs (i. e., the SVTs) and it is in the responsibility of the SVTs to respond to requests of the SUT. Furthermore, SVTs recurrently send to the BSCs the last valid actuator control values that have been received from the SUT.

While a controlled access eases the design of the SVT and supports replicable results at least in the time domain, the arbitrary access scheme is more flexible since it supports any SUT without requiring knowledge about its internal timing. However, we must take assumptions on the maximum change rate of the environment variables consumed and manipulated by the SUT, since the SVT must be fast enough to keep up with changes in the environment. The environment variables must be communicated to the SVTs at least with the Nyquist rate [Nyq28].

4.3.3 Types of SVTs

We distinguish between two types of SVTs, namely *sensor SVTs* that mimic the behavior of a sensor and *actuator SVTs* that mimic the behavior of an actuator.

Physical sensor devices can offer sensor data in the value and/or in the time domain. Both kinds of sensors can be emulated by a sensor SVT. An example of a sensor that delivers sensor data in the value domain is an infrared distance sensor. The processing unit that interfaces the infrared distance sensor receives an analog signal from the sensor reflecting the measured distance. An example of a sensor that delivers sensor data in the time domain is an ultrasonic sensor. An ultrasonic sensor is triggered by a processing unit to send an ultrasonic ping. As soon as this acoustic signal is echoed back to the sensor, the sensor informs the processing unit about the reception of the ping. The processing unit calculates the time from sending the signal until the reception of the signal in order to get a distance measurement.

Similar to sensor devices, physical actuator devices require distinguishing between value and time domain. There are many different devices that are controlled by the setting of an analog value (e. g., LEDs, simple electric motors). A time-dependent actuator SVT is for instance used in the case of a PWM signal that drives an actuator.

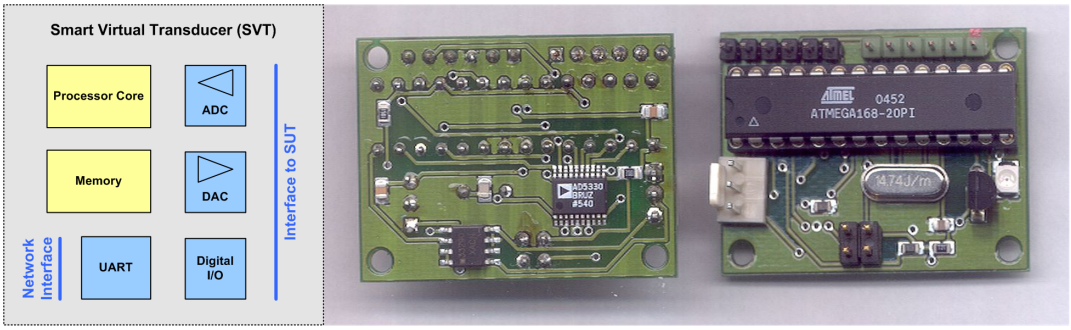

Figure 4.10: Block diagram and prototype of an SVT with D/A converter

4.3.4 Prototypical Realization

As depicted in figure 4.10, an SVT consists of a processor core, memory, a UART, as well as digital and analog I/O. The prototype given in figure 4.10 includes an Atmel ATmega168 micro controller [Atm04] and an Analog Devices 8-Bit digital-analog converter (AD5330) [AD00].

The ATmega168 is a low-power 8-bit microcontroller that is based on a RISC architecture. According to [Atm04], the ATmega168 achieves close to 1 MIPS per MHz. The ATmega168 offers 16 Kbytes of in-system programmable flash, 512 bytes EEPROM, as well as 1Kbyte SRAM. It operates on 23 general purpose I/O lines and 32 general purpose working registers. The ATmega168 involves three timers / counters with compare modes, internal and external interrupts, serial programmable USART, a byte oriented 2-wire serial interface, an SPI serial port and a programmable watchdog timer.

The AD5330 is an 8-bit digital-analog converter that operates from a 2.5 V to 5.5 V supply. According to [AD00], the power consumption is only 115 μA at 3 V or 80 nA in power down mode. The AD5330 offers a double-buffered parallel input logic and an output range of 0 - V_{ref}.

4.4 Revising the Rationale

In this section we revise the rationale for a generic distributed HiL simulation framework as given in section 4.1 with respect to the proposed solution discussed in sections 4.2 and 4.3. Thus, we discuss properties of the HiL simulation framework with respect to the ability to establish deterministic interaction between the HiL simulation system and the SUT. Furthermore, we show that the proposed solution supports a scalable distributed HiL simulator interfacing an SUT across arbitrary transducer interfaces and offering means for monitoring and on-line configuration.

4.4.1 Deterministic Interaction

As mentioned earlier, reproducibility of a simulation run requires deterministic interaction between the HiL simulator and the SUT. Therefore, the HiL simulator must

share a common time base with the SUT and both, the SUT and the HiL simulator must not exhibit intrinsic sources of indeterminism, e. g., by suffering from race conditions.

The presented HiL simulator shares its existing global time base with the SUT. Regarding the HiL simulator, deterministic behavior can be established due to the usage of a time-triggered communication and execution scheme. Deterministic construction of the SUT lies outside the sphere of control of the HiL simulator and requires a deterministic architecture. In case a time-triggered architecture is used for the SUT, sources of indeterminism can be avoided by design.

4.4.2 Distributed HiL Simulation System

An HiL simulator as outlined in section 4.2, consists of nearly-independent node computers (i. e., FSCs, BSCs) that are precisely specified at the level of their linking interfaces [KS03]. Time-triggered communication between the simulation components enables composability which is an important prerequisite for the simultaneous development of these components.

Furthermore, the presented approach is scalable in the sense that a given HiL simulator can be extended by further BSCs or FSCs. Due to the property of composability, it is possible to integrate several HiL simulators to form one single (large) HiL simulator. In the case of a large number of employed components that reaches beyond the limitations of TTP/A (max 255 nodes), a multi-cluster HiL simulator can be set-up, establishing the interaction between different clusters via gateways.

The nodes of an HiL simulator are physically interconnected via a digital serial interface, thereby supporting the application of HiL simulation to an SUT with remote transducers. Furthermore, our approach aims at complexity reduction regarding the set-up of a simulation for large distributed real-time systems. Each node of the HiL simulator implements a well-defined self-contained function, like for instance the interaction with one particular transducer or the execution of a particular part of the simulation model. Hence, an HiL simulator can be constructed out of modular components on the basis of well-defined interfaces.

Our proposed approach involves multiple hardware components, each consisting of a CPU, memory, and physical I/O. Thus, a multirate HiL simulation can be realized (refer to section 3.1) targeting at fast interaction with a particular physical transducer (by an FSC) while at the same time supporting a comparably slower and potentially resource intensive calculation of the simulation model (by a BSC).

4.4.3 Arbitrary Transducer Interfaces

The proposed concept of an SVT as outlined in section 4.3 supports the interaction of an HiL simulator with the SUT across an arbitrary transducer interface. Thereby, the concept of SVT supports the separation of concerns and thus leads to a reduction of the cognitive complexity when setting up an HiL simulation system. The simulation model, executed at the BSCs, is decoupled from the specific SVT elements that

interact with the SUT. Thus, it is not necessary to include the behavior of a particular transducer element in those parts of the simulation model that are executed at the BSCs. Changes of transducer elements (e. g., upgrade of a transducer to a newer model) do not directly influence the BSCs because the behavior models of the transducers are confined by the SVTs.

The SVT approach supports reusability by the provision of configurable FSCs. Since the implementation of an SVT mainly depends on the transducer that is replaced, an SVT can be reused in other applications whenever the same kind of transducer is employed. Although features like the frequency of the update value and smoothing parameters depend on the control environment, these functions can be generically implemented and parametrized for a particular application.

The approach is open to any kind of "black box", i. e., any transducer-specific interface can be implemented on an SVT. Thus, integration tests, both open loop and closed loop, of a wide variety of SUT configurations can be performed with this approach. Furthermore, it is possible to use an SVT as a connection system that resolves property mismatches between a given simulation model and a particular legacy I/O board (as depicted in figure 4.3).

4.4.4 Monitoring and Configuration

As discussed in section 4.1, an HiL simulator should offer means for monitoring and run-time configuration. Neither monitoring nor run-time configuration must introduce a probe effect on the real-time behavior of the HiL simulator. Our proposed approach targets at these requirements and supports non-intrusive monitoring as well as a mechanism for run-time configuration that does not interfere with the real-time interaction of the HiL simulator components.

For observing the behavior of the HiL simulation system we propose to use a gateway BSC [Sch03] that captures relevant real-time images (i. e., simulation data) transmitted across the LLI and forwards these real-time images to an HMI computer. Alternatively, a BSC can store transmitted data during a simulation run.

On-line configuration of the HiL simulator is based on the concept of master-slave rounds of the TTP/A protocol. Thereby, the master reads/writes a particular element of the IFS of each TTP/A node. Master-slave rounds are triggered by the master between multi-partner rounds. Hence, we propose the realization of a gateway BSC that connects to an HMI computer and that forwards requests of the operator directly to the relevant simulation component via the master node. Physically, a gateway BSC can also be directly implemented at the master node.

4.5 Chapter Summary

In this chapter we investigated the rationale for a scalable, component-based, deterministic HiL simulation system framework based on the TTA. A major requirement of such a framework lies in deterministic interaction between HiL simulator and SUT

across well-specified interfaces. A distributed HiL simulator enables parallel development, scalability, complexity management, and multirate simulation. An HiL simulator shall provide support for arbitrary transducer interfaces, mechanisms for monitoring, run-time configuration, and data logging.

The proposed HiL simulation framework involves a set of Backend Simulation Components (BSCs) and Frontend Simulation Components (FSCs). An FSC controls the physical interconnection between the HiL simulator and the SUT and consists of a digital interface to the TTP/A network and a second interface that resembles the COI of the SUT. This second interface is either a digital fieldbus interface (fieldbus FSC), an arbitrary transducer interface (virtual transducer FSC), or a physical transducer (real transducer FSC). A BSC is used to execute part of a distributed simulation model. It has a digital interface to the TTP/A network (simple BSC) and (optionally) a second digital gateway interface to connect to an external system (gateway BSC). A virtual transducer FSC can be realized by an SVT, which emulates a physical sensor or actuator element. An SVT consists of a processing unit, a communication interface, and a transducer specific interface.

In the proposed HiL simulation framework, the time base of the HiL simulator is synchronized to the time base of the SUT which is a prerequisite for deterministic interaction between the HiL simulator and the SUT. The presented concept supports composable construction of a distributed HiL simulator based on nearly-independent FSCs and BSCs. The HiL simulator supports non-intrusive monitoring and on-line configuration enabling simulation runs without probe effect on the SUT.

Chapter 5

Testing of an Integrated System

In the previous chapter, we discussed the constituting elements of a novel HiL simulation framework based on time-triggered execution and communication. In this chapter we present the adoption of this framework for the purpose of testing integrated systems. Furthermore, we investigate a stepwise X-in-the-Loop simulation process including MiL, SiL, and HiL simulation. As part of SiL testing, we present a pre-validation framework for virtual integration of integrated systems. Substantial parts of the chapter origin from [SOE07, OS07].

5.1 Integrated Architecture

The increasing number of electronic functions in the automotive, aerospace, or industrial control domains requires bundling and seamless integration of application subsystems from different vendors on the same hardware platform. A study of A.D. Little [BE05], for instance, states that currently about 70 ECUs are used in cars and in the next decade, a significant reduction to about 20 ECUs is expected. Application subsystems may have different levels of criticality and must be temporally and spatially protected against each other in order to avoid the development of each module on the shared hardware according to the highest safety requirements.

Targeting at these challenges, so-called integrated architectures are developed that provide means to handle the complexity of distributed applications while supporting efficient integration of functions into the shared hardware. The DECOS integrated architecture [KOPS04, OPHES06] is an example for an integrated system architecture which builds upon the validated architectural services of a time-triggered core architecture. DECOS is an integrated project within the Sixth Framework Program of the European Commission and targets at a European cross-industry approach for integrating multiple application subsystems within a single, distributed computer system. Other well-known integrated architectures are AUTOSAR [AUT06b] (in the automotive domain) and IMA [ARI91] (in the aerospace domain).

5.1.1 System Structure

Many large applications (e. g., in the automotive or aerospace domains) consist of a number of nearly independent application subsystems, each of them providing a major part of the overall application. We call such an application subsystem a Distributed Application Subsystem (DAS). In the automotive domain, the powertrain subsystem, the comfort subsystem, and the multimedia subsystem are examples for DASs. Examples of DASs in a present-day avionic application are the cabin pressurization system, the fly-by-wire system, and the in-flight entertainment system. A DAS is composed of smaller functional elements called *jobs*. *A job is the basic unit of work that employs the communication system for exchanging information with other jobs, thus working towards a collective goal* [KOPS04].

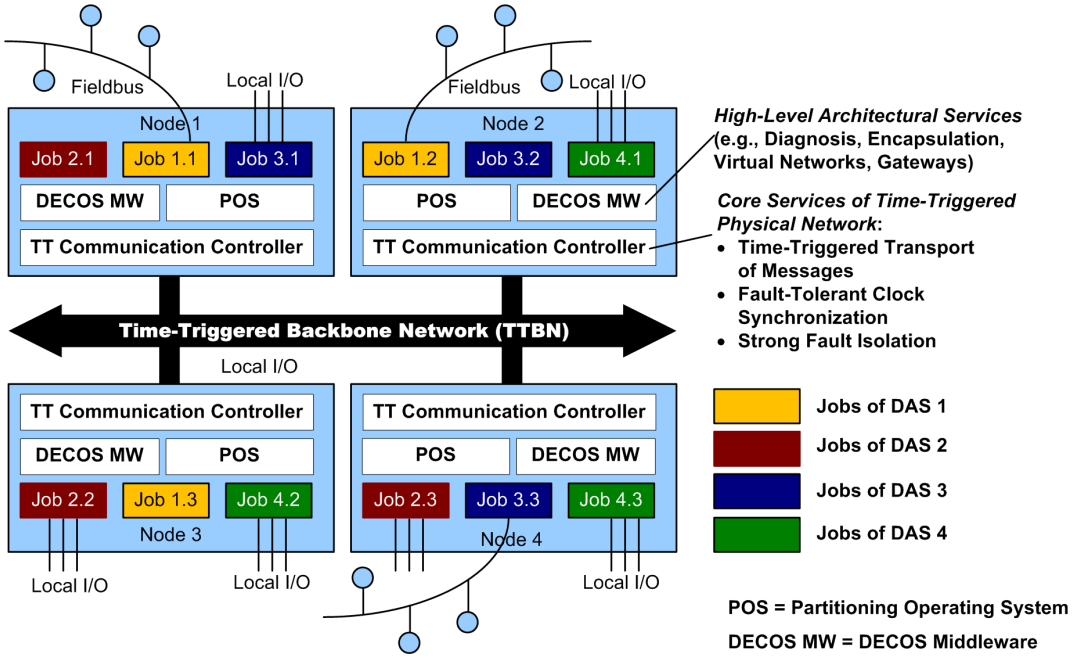

Figure 5.1: Distributed system in the DECOS system architecture

In an integrated architecture a single distributed computer system serves as the execution platform for multiple DASs. As depicted in figure 5.1, each node computer of the distributed computer system contains jobs of one or more DASs. Likewise, the communication network that interconnects the node computers serves the transport of messages between jobs of more than one DAS.

In the following, we discuss the structural elements of the DECOS architecture (i. e., network, nodes, environment), because this system architecture is used for the construction of the framework for SiL and HiL simulation.

5.1.2 Communication Network

Physically, the real-time computer system in the DECOS architecture consists of a set of nodes that are interconnected by a *Time-Triggered Backbone Network (TTBN)*.

Prototype implementations of the DECOS architecture are available with different time-triggered communication protocols used for the TTBN (e. g., TTP [TTA02], FlexRay [FX05]). The rationale behind choosing a time-triggered communication protocol is in the requirements with respect to predictability and fault-tolerance in safety-critical applications [Rus01] and the suitability for ultra-dependable systems [SWH95]. Time-triggered communication protocols are characterized by a guaranteed message transport with low jitter, error containment between node computers, and a fault-tolerant distributed global clock service.

5.1.3 Node Computers

A node computer provides an execution environment for multiple collocated jobs of one or more DASs as shown in figure 5.1. Each job implements a part of the application functionality and is within the responsibility of a single organizational entity (e. g., a specific supplier).

The allocation of computational resources (e. g., memory, CPU time) to jobs occurs using a partitioning operating system with support for fault isolation and modular certification [SHW+06, HPOES05]. The partitioning operating system implements mechanisms for spatial and temporal partitioning in order to encapsulate the individual jobs. The scheduling of jobs needs to ensure that a timing failure of a job, such as a worst-case execution time violation, does not affect the CPU time available to other jobs. In analogy, the spatial partitioning mechanisms of the partitioning operating system enforce memory protection between jobs (e. g., with a memory management unit).

The interaction with other jobs occurs through the services provided by the DECOS middleware. The DECOS middleware offers high-level architectural services, which serve as a baseline for the development of applications. These services constitute the interface for the jobs to the underlying platform. Among the high-level services are gateway services, virtual network services, encapsulation services, and error detection services. On top of the time-triggered physical network, different kinds of virtual networks are established as overlay networks [OP05] and each type of virtual network can exhibit multiple instantiations. The access point of a job to the virtual network is denoted as a *port*.

Gateway services selectively redirect messages between virtual networks and resolve differences with respect to operational properties and naming. The encapsulation services control the visibility of exchanged messages and ensure spatial and temporal partitioning for virtual networks in order to obtain error containment.

Below the DECOS middleware, each node computer in figure 5.1 contains the communication controller. The communication controller executes a time-triggered communication protocol (i. e., the TTBN) as required for accessing the network. It provides so-called core architectural services (i. e., time-triggered transport of messages, fault-tolerant clock synchronization, strong fault isolation), which are used as the basis for the implementation of the high-level architectural services in the DECOS middleware.

The rationale for distinguishing between the core architectural services and the high-level architectural services is the ability to exploit existing time-triggered communication protocols (e. g., TTP) for the construction of an integrated architecture. TTP is a prominent candidate for it has been demonstrated by formal analysis [Rus02] and experiments [ASBT03] that TTP is appropriate for the implementation of applications in the highest criticality class in the aerospace domain according to RTCA DO-178 B Level A.

5.1.4 Environment

The coupling of an integrated system with its environment is established via transducers. Transducers can either be connected directly to the integrated system or interfaced via a fieldbus. The latter approach simplifies the installation from a logical and a physical point of view and is extendable but might introduce increased latency of sensory information and actuator control values because the latency of the fieldbus (i. e., the time that is required for capturing, processing, and sending a sensor value to the integrated system via the fieldbus) adds to the total signal path.

Every job has access to its relevant transducers, either directly via the controlled object interface or via a virtual network of known temporal properties. Hence, an HiL test procedure requires finding adequate interfaces between the HiL simulator and the integrated system.

5.2 X-in-the-Loop Testing

The verification of an integrated system starts with module tests (refer to figure 5.2). Module tests relate to the process of verifying jobs (or even smaller elements like tasks). In a further step, single DASs are tested, including jobs of the DAS and virtual networks that connect these jobs. After that, system tests are performed with multiple DASs.

In the case of model-based development of the integrated system based on MATLAB/Simulink models, first tests are performed in the MATLAB/Simulink environment. Hence, an MiL simulation of jobs or even of whole DASs is done in MATLAB/Simulink. Focusing on the temporal interrelationship of different jobs and DASs, SiL tests are conducted after MiL tests have shown a sufficient degree of maturity of the DASs. As a final verification step, HiL tests are conducted.

In the following, we discuss X-in-the-loop techniques, namely MiL, SiL, and HiL with respect to their application to integrated systems.

5.2.1 Model-in-the-Loop (MiL)

MiL corresponds to the process of testing a single computational unit, i. e., a job or parts thereof, with a simulated system model. A job can be tested by an MiL test as soon as a functional model of the job is available. Typically, an MiL test is directly

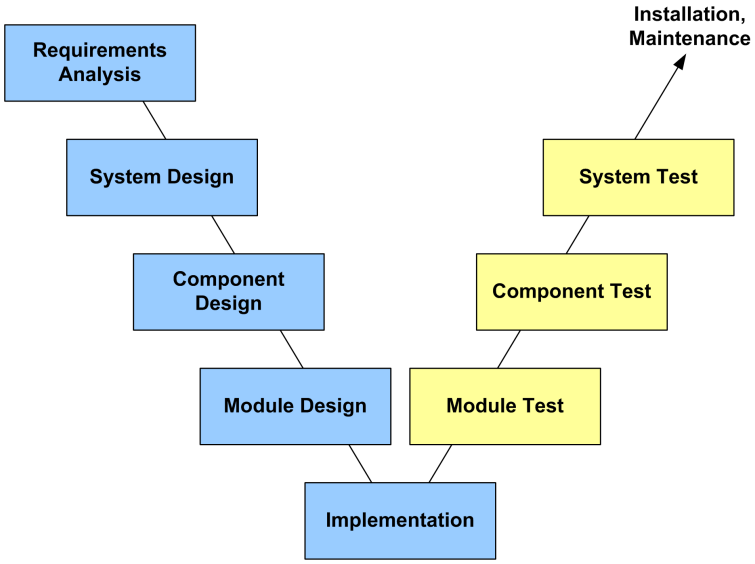

Figure 5.2: V-model

conducted within the development environment, e. g., within MATLAB/Simulink. MiL tests can easily be set-up by defining certain input test cases that are provided to the model of a job in the model-based environment. Timing properties are usually neglected in an MiL test.

5.2.2 Software-in-the-Loop (SiL)

SiL testing is a pre-validation technique, where executable code is tested in a simulated (software) environment that is usually implemented on a desktop computer. In the automotive domain, the ambition to virtually integrate software modules on a simulated platform led to the notion of *virtual integration platforms*. In [GFL+02], it is proposed that *the integration should take place on a virtual platform that consists of models of the hardware and software components that constitute the building blocks of the overall model of the distributed function and architecture.*

In order to analyse the behavior of correlating DASs of an integrated architecture by an SiL, we propose the use of a simulation framework as presented in [OS07]. Such a simulation framework combines (1) a simulation of virtual networks, (2) a simulation of time-triggered partitioning operating systems, (3) an environmental simulation, and (4) application code created from models of application functionality.

Using this simulation framework, developers can determine whether application subsystems operate correctly in the integrated architecture. The aim of the simulation framework is to guarantee the temporal ordering of events during the simulation. However, the simulation framework does not support real-time execution of the SiL test. Both, the SUT and the environment of the SUT are simulated. The simulation of an integrated system can be executed on a single desktop computer as discussed in [OS07].

5.2.3 Hardware-in-the-Loop (HiL)

HiL tests allow to observe the actual influence of the hardware characteristics of the SUT including the physical behavior (timing!) of hardware signals and execution of the SUT in real time. Thus, testing with an HiL simulation is closer to the real application than a pure software simulation.

HiL tests of an integrated system must preserve the prerequisites for achieving reproducible test results (refer to chapter 4). Hence, we focus on deterministic interaction between an HiL simulator and an integrated system at the physical interface of this integrated system. This allows for non-intrusive (black box) tests. We call an integrated system that is tested by an HiL simulation an Integrated System-Under-Test (ISUT).

5.3 Virtual Integration

As mentioned in section 5.2, virtual integration deals with SiL simulation for the purpose of pre-validating an integrated system. A virtual integration framework for an integrated system involves:

- **A framework implementation** enabling developers to simulate the behavior of application subsystems that share computational and communicational resources with other application subsystems based on time-triggered schedules.
- **The combination of virtual integration with environmental simulation** in order to support environmental simulation in the absence of the physical environment for testing real-time applications by means of simulation.
- **Tool-support for virtual integration on the framework implementation** that starts with a model based development of the application behavior, the environment, and the DECOS execution platform. These models (in our case study we used MATLAB/Simulink) are automatically transformed into input for simulation tools.

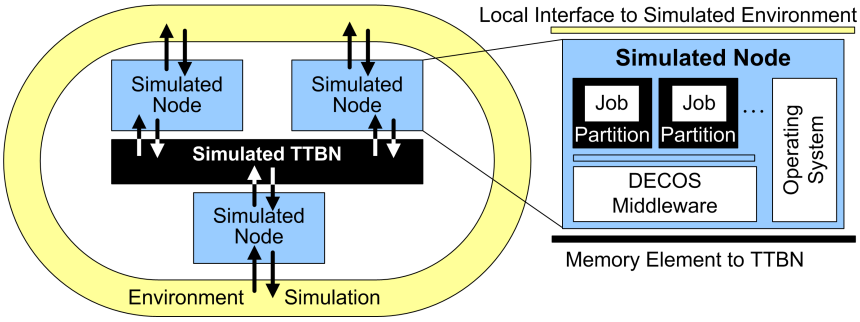

Figure 5.3: System model of the simulated real-time system

5.3.1 Structure of Virtual Integration Framework

The framework implementation of the virtual integration platform supports the simulation of a distributed computer system, complying to the DECOS architecture, along with the environment. As depicted in Figure 5.3, the simulated distributed computer system consists of a set of simulated nodes, each hosting jobs of one or more DASs. The simulated TTBN is a substitute for a physical network interconnecting these nodes. The simulation of the environment allows jobs to interact with (simulated) sensors and actuators.

Time-Triggered Backbone Network

This part of the virtual integration platform simulates a network with a time-triggered communication protocol (e. g., TTP [TTA02] or FlexRay [FX05]). The TTBN executes TDMA and divides time into slots that are statically assigned to the nodes. A particular slot is used by one node to broadcast a message, while all other nodes receive the message. A periodically recurring sequence of slots that enables each node to send a message is called a communication round.

The simulated TTBN provides to the simulated nodes a time-triggered message transport service, which performs periodic exchanges of state messages at predefined global points in time. At each simulated node, the interface to the TTBN is a memory element. The memory element contains outgoing state messages that are written by the application in the simulated node and read by the simulated TTBN prior to broadcasting them. In addition, the memory element contains incoming state messages that are read by the application in the simulated node and updated by the simulated TTBN with state messages received from other nodes.

In a physical TTBN, this memory element would be provided by a communication controller at each node, e. g., TTP controller C2NF [TTC06] or FlexRay Controller MFR4200 [Fre05]. This memory element, which is called CNI in TTP and Controller Host Interface (CHI) in FlexRay, is provided by most time-triggered communication protocols with syntactic differences of state messages (e. g., header format) and protocol-specific constraints (e. g., only one message sent by a node per communication round in TTP [TTA02], same size for all state messages in FlexRay [FX05]).

Simulated Node

A simulated node contains *partitions*, each of which is an encapsulated execution space within a node with *a priori* assigned computational resources (e. g., CPU, memory). Partitions are the target of job allocation and each job is always assigned in its entirety onto a partition, i. e., a job is never fragmented onto multiple partitions.

A simulated node also contains the DECOS middleware for the implementation of the generic architectural services (i. e., virtual networks, gateways). The DECOS middleware maps the memory element provided by the TTBN to ports that serve as the access points to the virtual networks. In order to enable jobs of a DAS to exchange information, the middleware layer provides to each job one or more *state*

ports and *event ports*. A state port contains a state variable that is accessed by the DECOS middleware at *a priori* specified global points in time. The state variable is either written by the DECOS middleware (data received from a virtual network and destined to an input state port) or read by the DECOS middleware (data from an output state port and destined to a virtual network). An event port, on the other hand, contains a message queue into which messages that are received from a virtual network are inserted by the DECOS middleware in case of an input event port. For an output event port, the DECOS middleware retrieves messages from the queue and sends them via the corresponding virtual network.

Environment

Environmental simulation resembles the physical surrounding of an integrated system. In a real system, a transducer would manage a set of real-time entities of the environment (e. g., speed of a motor). Each time a simulated node interacts with the environmental simulation during virtual integration, it accesses a real-time entity of a simulated transducer element.

5.3.2 Inputs to the Virtual Integration Framework

The virtual integration platform takes as an input three simulation models (refer to figure 5.4): (1) a simulation model of the application, (2) a simulation model of the distributed execution platform, and (3) a simulation model of the environment. Using code generation tools, we produce for each model a set of software modules that can be executed within the simulation framework. The simulation framework provides feedback to the designer concerning the behavior of the application on a given virtual integration platform which is defined via the models of the execution platform and the environment.

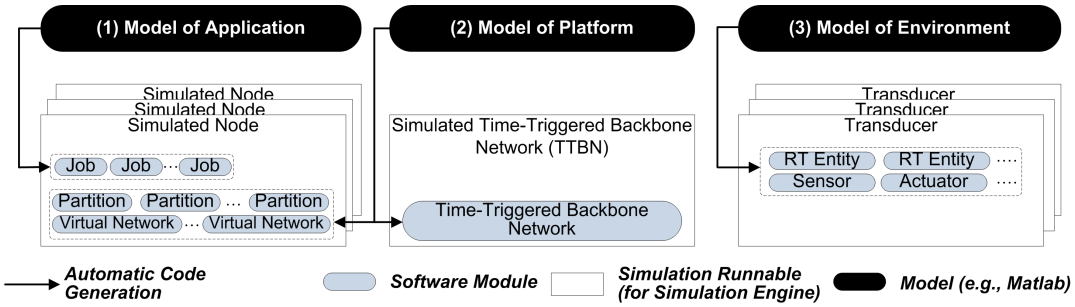

Figure 5.4: Model-based integration on virtual integration platform

Simulation Model of Application

The simulation model of the application decomposes the overall system (e. g., onboard electronic system of a car). For specifying this model, a suitable formalism can be selected that fits to the respective application domain (e. g., MATLAB/Simulink).

The simulation model of the application defines for each job a corresponding data transformation. Such a data transformation takes as an input the messages that are provided to the job, the information acquired from the sensors (i. e., at the interface to the environment), and the internal state of the job. The outputs of the transformation are messages generated by the job, information for the actuators (i. e., to the environment), and updates of the internal state of the job.

Distributed Execution Platform

The model for the distributed execution platform (virtual integration platform) describes the distributed platform, on which DASs shall be executed. The distributed execution platform encompasses the following three parts:

Partitions of Execution Environment: The simulation framework supports a time-triggered operating system, which associates each partition to a periodically recurring time slice. Consequently, a partition is characterized by the temporal attributes of the time slice (i. e., start and period of invocation, duration of execution), as well as by the spatial attributes (i. e., location of the partition expressed as a particular node and processor within the node).

Virtual Networks and Gateways: As part of the platform model, the configuration of the virtual networks and gateways is defined. The configuration of the virtual networks provides information on the types of available virtual networks along with the control paradigm (i. e., time-triggered or event-triggered), and the communication topology (e. g., broadcast, point-to-point). Also, for each virtual network the mapping to the underlying TTBN is specified by reserving TDMA slots at the time-triggered physical network. For the virtual gateways, the platform model defines which messages shall be redirected between virtual networks.

Time-Triggered Backbone Network (TTBN): The third part of the platform model captures the physical network of the platform. In particular, the TDMA scheme of the TTBN is laid down. In conjunction with the virtual network configuration, this TDMA scheme determines the temporal properties (i. e., bandwidth, latencies) of the virtual networks.

Environment Model

The model of the environment is derived from an analysis of the natural environment and the used sensors/actuators. In analogy to the application model, a proper formalism has to be selected to specify the environment model. In our work we used MATLAB/Simulink for modeling the environment of the integrated system.

5.3.3 Simulation on Virtual Integration Platform

Using automatic code generation, the application-, platform-, and environment models are transformed into code that can be executed on a Linux-based PC in conjunction with a *simulation engine*. The Real-Time Workshop processes the MATLAB/Simulink models for the application and the environment, thus producing software modules for jobs and transducers. For the platform, we use code generation tools that have been introduced in previous work [OH06, HHBC06] for the deployment on a physical target system.

We denote a task, which executes generated code of the simulation model, as a *runnable*. The simulation framework employs three types of runnables: a *TTBN runnable* for simulating a physical time-triggered network, *node runnables* for the simulated nodes, and *transducer runnables* for simulating the environment (refer to figure 5.4).

The purpose of the simulation engine is the establishment of a binding between the runnables (refer to figure 5.5). The simulation engine synchronizes the execution of the runnables by enforcing a time-triggered action lattice for the computational activities of the simulated nodes, the communication activities of the simulated TTBN, and the state changes in the environment. In addition, the simulation engine provides mechanisms for the information exchange between runnables. The interaction between runnables takes place via the following shared memory regions:

- **Communication Network Interface (CNI):** The CNI links the node runnables of the simulation framework. In a real system, the CNI would be realized as a dual ported memory that is accessible by both, the application of the node and the communication controller (refer to figure 5.6).
- **Controlled Object Interface (COI):** The COI represents the interface between the integrated system and its environment. In a real system this interface would be realized through the physical interaction of transducers with their environment.
- **Simulation System Interface (SSI):** The SSI is used for the interaction between transducer runnables. The SSI is used to exchange messages not being visible at the COI.

In the following, fundamental properties of simulation engine and runnables are outlined, including their respective interfaces.

Simulation Engine

The simulation engine manages the activation of runnables at pre-defined instants. Hence, it provides a (logical) simulation time that emulates a physical clock. Based on the progression of the simulation time (i. e., increment of an internal counter variable), the simulation engine processes a deterministic, round-based schedule. The

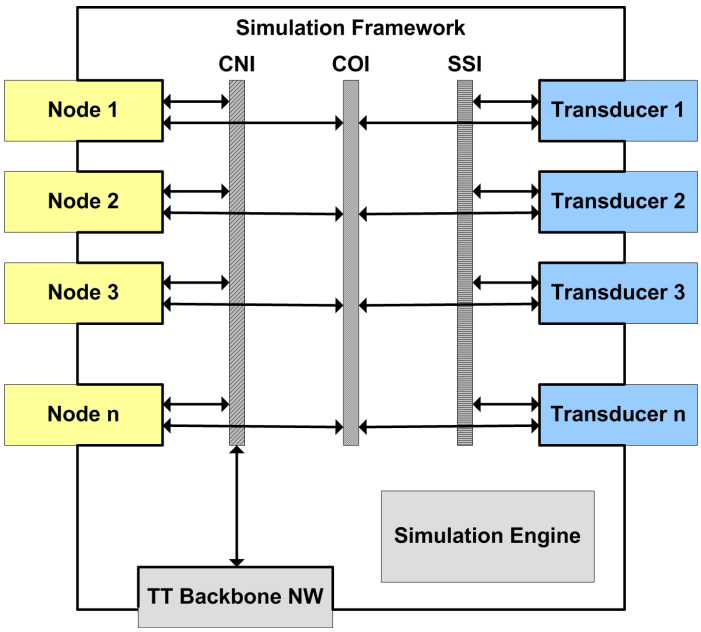

Figure 5.5: Simulation framework

smallest entity of this schedule is a slot. Within each slot the simulation engine can activate either one or several runnables.

The activation of the involved runnables is realized with semaphores whereas a pair of semaphores is related to each runnable. Hence, the simulation engine starts a certain runnable by performing a V operation on the first semaphore and waits for the runnable to finish by performing a P operation on the second semaphore.

Runnables are statically linked with the simulation engine. Thus, the identifier of a certain runnable, i. e., the pair of semaphores, as well as the instants of activation of this runnable are defined within the simulation engine. The dynamic behavior of the runnable is given within the particular runnable, and is therefore not part of the simulation engine.

TTBN Runnable

The TTBN runnable emulates a physical TTBN that connects the nodes of an integrated architecture. Hence, the TTBN runnable enables the interaction of node runnables across the CNI. Therefore, it operates on the CNI and periodically (i. e., each time it is activated by the simulation engine) updates the CNI according to a statically defined Message Descriptor List (MEDL).

In order to better explain the purpose of the TTBN runnable it should be mentioned that each node runnable has access to a particular region of the CNI (i. e., a particular shared memory region). Each node runnable can read/write messages from/to this CNI region. Each time, the TTBN runnable is activated, it copies messages that have been written to a certain CNI regions by a node to all other CNI regions of all other nodes.

In a real (i. e., non-simulated) integrated system, this task would be performed by communication controllers that send messages from a particular node, receive messages from all other nodes, and perform the respective CNI read/write operations.

Node Runnables

As depicted in figure 5.5, the simulation framework involves a set of node runnables that correspond to the application behavior of the integrated system. Each node runnable represents the dynamic behavior of an integrated node. Hence, each node runnable consists of a set of jobs that are executed according to a time-triggered schedule. The deterministic activation of jobs by a node corresponds to the activation of partitions by a time-triggered partitioning operating system.

As explained in the previous subsection, the interaction between different node runnables happens across the CNI. Furthermore, the interaction between node runnables and the environment is realized via the COI. The COI is realized as a separate shared memory region. This shared memory region consists of messages that reflect information exchanged with transducers. For instance, the current analog value of an infra-red distance sensor would be stored within an integer variable *distance_analog* as part of the COI. The involved transducer runnables (refer to the next subsection) are responsible to periodically update the COI.

We propose the development of the dynamic behavior of node runnables with MATLAB/Simulink. After a MATLAB/Simulink model has been realized, it is possible to perform preliminary tests within MATLAB/Simulink. After that, C code of the model is automatically generated with the Real-Time Workshop Embedded Coder and this C code is included in the node runnable.

The simulation framework provides wrapper code that basically performs three steps each time a node runnable is activated by the simulation engine:

- **Read input:** All required input data from CNI and COI is read by the node runnable. This input data is used to set the input variables of the MATLAB/Simulink model.
- **Execute model:** One step of the MATLAB/Simulink model is executed by calling the respective function that is provided by the automatically generated code.
- **Write output:** The output variables of the MATLAB/Simulink model are read and this data is written to the according CNI and COI regions.

The integration step that links a particular node runnable wrapper to the MATLAB/Simulink model has to be taken only once. If the respective MATLAB/Simulink model is modified (e. g., for calibration purposes), the node runnable is automatically updated after generating code for this updated model.

Transducer Runnables

The environment of the integrated system is simulated by the use of transducer runnables. Each transducer runnable maintains a set of real-time entities of the environment. The interaction between the node runnables and the transducer runnables takes place via the COI.

Interaction between different transducer runnables takes place across the SSI – again a separate shared memory region. The SSI is used to exchange messages of a distributed environmental simulation which are not visible at the COI. For example, a simple application could involve two transducer runnables. We assume that the first transducer runnable models the dynamic behavior of an electric motor that is turned on and off by a node runnable. A second transducer runnable models a sensor capturing the current speed of the electric motor and providing speed information to another node runnable. In such an application, a value representing the current speed of the electric motor is written to the SSI by the first transducer runnable. This value is read and further processed by the second transducer runnable.

The structure of a transducer runnable is quite similar to the structure of a node runnable. Hence, each time a transducer runnable is activated data is read from the COI and from the SSI, a transducer model is executed, and data is written to the COI and to the SSI. As for the development of node runnables, we propose a model based approach (with MATLAB/Simulink) for the development of the transducer runnables. The simulation framework therefore provides wrapper code for the involvement of automatically generated code of transducer models.

5.4 HiL Testing

In the following we discuss the applicability of the HiL simulation framework as presented in chapter 4 for conducting HiL tests with an integrated system. It should be mentioned that due to its generic interaction mechanism (i. e., support of arbitrary transducer interfaces) the simulation framework is well-suited to support HiL testing of a large range of integrated systems. In the following we investigate the particularities of an integrated architecture with respect to an HiL simulation. Furthermore, we present the application of the HiL simulation framework to an exemplary automotive system.

5.4.1 HiL Simulation Framework

As with SiL simulation, HiL simulation of an integrated system involves a simulation of the environment of the integrated system. Thus, the environmental simulation model from the SiL simulation can be reused when setting up an HiL simulation system. In contrast to an SiL simulation, an HiL simulation involves the real-time execution of this environment model. Thus, the WCET of the execution of a simulation step of the environment model must be considered.

Based on the general definition of an HiL simulation system as outlined in sec-

tion 3.1, an HiL simulation system for an integrated system consists of the following two building blocks:

Integrated System-Under-Test (ISUT): The ISUT is either an integrated system as depicted in figure 5.1 or a part thereof. The ISUT interacts with its environment across the COI.

HiL Simulator: The aim of the HiL simulator is to substitute the environment or parts of the environment of the ISUT. The interfaces of an HiL simulator that are relevant for the interaction with the ISUT must resemble the interfaces of the ISUT in the temporal and functional domains. The HiL simulator executes a simulation model of the process under control of the ISUT and a model of the transducers. The input and output from these models is exchanges with the ISUT across the COI.

The COI, linking the ISUT and the HiL simulator, can either be a standardized digital transducer interface or an arbitrary transducer-specific interface (e. g., an analog interface). Deterministic interaction between the HiL simulator and the ISUT across the COI is an essential aspect for any kind of interface.

By applying the HiL simulation framework according to the architectural description given in section 4.2, we involve a distributed HiL simulator consisting of a set of FSCs, a set of BSCs, and a master node. The master node is part of the HiL simulator, i. e., it triggers the individual BSCs and FSCs according to a pre-defined schedule. Furthermore, the master is a (passive) member of the ISUT, i. e., it synchronizes its time-base with the time-base of the ISUT. Hence, the master establishes synchronism between the ISUT and the HiL simulator without effecting the ISUT's execution.

As depicted in figure 5.6, the interaction of nodes of an integrated system with their environment is realized via an arbitrary transducer interface. This interaction includes value/time-dependent analog and/or digital direct I/O as well as standardized fieldbus interfaces. An FSC connects to nodes of the integrated system for the purpose of interacting with these nodes across a particular transducer interface. FSCs and BSCs collectively execute the distributed simulation model of the environment of the integrated system.

5.4.2 HiL Simulation with an ISUT

HiL simulation of an integrated system supports incremental testing. Starting with a single integrated node, a stepwise inclusion of jobs of the integrated system in the HiL simulation is possible. At each step, the environment model of the real-time system is simulated (by BSCs) and the coupling between this simulation and the actual ISUT is established with FSCs. With separate FSCs it is possible to scale the HiL simulation from a small ISUT (e. g., a single integrated node with only one job) up to a complete integrated system by adding additional FSCs as required.

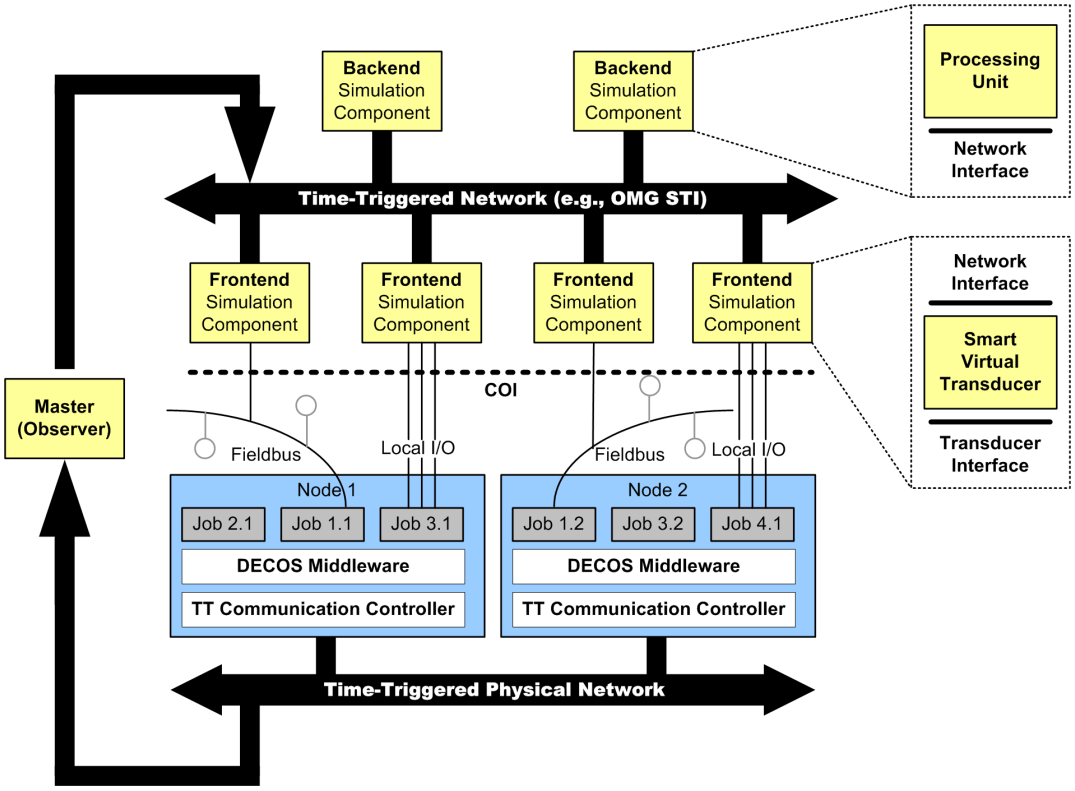

Figure 5.6: HiL simulation with an integrated system

Regarding the HiL simulation system for an integrated system, we can establish deterministic behavior due to the usage of a time-triggered communication and execution scheme. Furthermore, an ISUT that is based on the DECOS integrated architecture builds on a time-triggered architecture that avoids sources of indeterminism by design. Hence, a distributed environmental simulation system as described above, based on time-triggered communication, supports the construction of an HiL simulation system with guaranteed deterministic interaction between the HiL simulator and the ISUT.

5.4.3 Exemplary Application

In the following, we discuss HiL simulation with an exemplary (integrated) automotive system.

Exemplary Application Using the Integrated Architecture

The example used to point out the HiL simulation framework includes two automotive DASs (which are part of a larger automotive electronic system):

- **Multimedia DAS:** Car drivers are no longer satisfied with cars being simple means of transportation. For this reason, today's luxury cars contain multi-

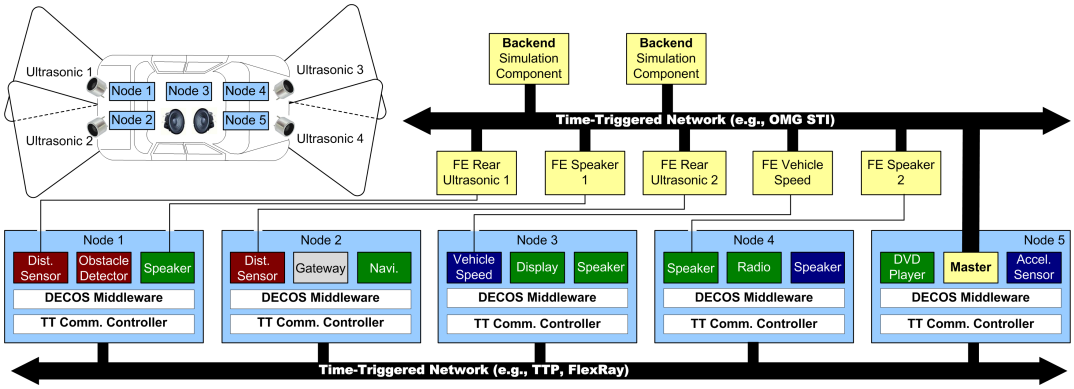

Figure 5.7: Exemplary integrated system with HiL simulation

media functionality such as DVD players, high-end audio systems, and GPS navigation systems. In addition, voice control and hands-free speaker phones relieve the driver of concentrating on multimedia devices instead of traffic.

- **Park assist DAS:** This DAS implements a parking aid with ultra-sonic sensors. In case a threshold for a minimum distance is exceeded, the DAS produces an acoustic alarm signal. Therefore, the park assist DAS encompasses four jobs reading inputs from ultra-sonic distance sensors. In addition, the DAS contains an obstacle detector job, which reads the distance measurements from the four other jobs and determines whether an alarm signal should be produced. In this case, the acoustic alarm signal is transferred via a gateway to the speaker jobs of the multimedia DAS.

Figure 5.7 depicts a possible implementation of these DASs using the DECOS architecture. Each node computer hosts multiple jobs, which can belong to different DASs (such as the multimedia or park assistant DAS).

HiL Elements for Exemplary Application

We assume that the above described example involves two kinds of transducers, namely ultra-sonic sensors for distance measurement of the park assist DAS and loudspeakers of the multimedia DAS. Hence, the interaction between the HiL simulator and the ISUT across the COI involves SVTs that emulate the behavior of an ultra-sonic sensor as well as SVTs that capture and process the signals provided by the audio system jobs of the ISUT.

As depicted in figure 5.7, the setup of the HiL simulation system additionally involves an FSC receiving the actual vehicle speed from the ISUT. Furthermore, a master node is required that controls operation of the involved SVTs and synchronizes the time-base of the HiL simulator to the time-base of the ISUT.

5.5 Chapter Summary

Integrated architectures (e. g., DECOS [KOPS04, OPHES06], AUTOSAR [AUT06b], IMA [ARI91]) provide means to handle cognitive complexity of large integrated systems by supporting efficient integration of functions. Thereby, mixed criticality application subsystems from different vendors are seamlessly integrated on top of the same hardware.

According to the terminology of the DECOS project, an integrated system consists of *Distributed Application Subsystems (DASs)* that are composed of *jobs*. *A job is the basic unit of work that employs the communication system for exchanging information with other jobs, thus working towards a collective goal* [KOPS04]. *Node computers* provide the execution environment for multiple collocated jobs of one or more DASs. The physical interconnection between node computers is established by a *Time-Triggered Backbone Network (TTBN)*. Integrated systems interact with their environment via transducers that are either connected directly to the integrated system or are accessed via a fieldbus.

X-in-the-Loop testing of an integrated system involves MiL, SiL, and HiL based testing. SiL tests can be performed by a *virtual integration platform* including the simulation of networks, operating systems, and the environment as well as application code created from models of the application functionality.

As soon as SiL tests are accomplished, HiL testing follows as a final pre-validation step. The HiL simulation framework presented in chapter 4 provides a generic interfacing mechanism by treating the *Integrated System-Under-Test (ISUT)* as a black box that is interfaced at the transducer level.

Chapter 6

Case Studies

During the work on this thesis, three different case studies have been set up. In this chapter the problem statement of each case study is briefly discussed. Additionally, essential elements are outlined with respect to the previously established terminology. Furthermore, the case studies implementations are described. Substantial parts of this chapter have been published in [Sch03, EPS04, ES04, SWE06, SEW06].

6.1 Digital Smart Transducer Gateway

6.1.1 Problem Statement

Rear distance measurement systems can be used in cars to aid the driver when backing into a parking space. In the course of this case study, a prototype of such a system has been developed, comprising three infra-red (IR) distance sensors, a display, and a sensor fusion controller (refer to figure 6.1).

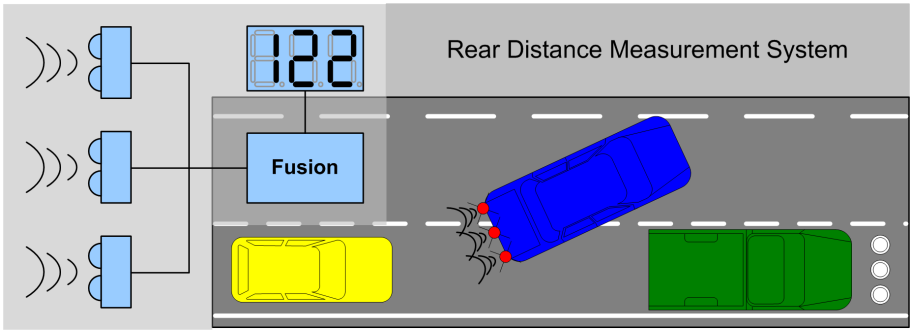

Figure 6.1: Rear distance measurement system

The development of the rear distance measurement system has been guided by the following development steps [ES04]:

- First, information about the process is gathered in order to formalize the process model.

- Second, the process and the physical transducers are modeled.
- Third, the real-time computer system is implemented and parameterized, e. g., calibration of sensor fusion algorithm.
- Fourth, HiL tests are applied in order to pre-validate the real-time computer system.
- Fifth, the system is connected to real transducers and tested in its physical environment.

The purpose of the case study has been to exemplify the first, second, third, and fourth development step. In the following we investigate on these development steps.

Process Environment

The distance measurement system is embodied in a car. Thus, the confidence of distance values received from the sensors depend on their conformance with the physical characteristics of a car. For instance, a vehicle is bound to a maximum acceleration value (positive/negative).

For the case study, a basic time continuous system is considered with state variables *(t, v, s, a)*, standing for time, velocity, space, and acceleration. Disruptive factors of the process environment like vibration, temperature, or ambient light have been neglected. Furthermore, the model has been discretized in order to allow discrete HiL simulation.

Assuming equably acceleration, the resulting formulas to model the physical process environment (i. e., movement of the vehicle) are as follows:

$$v = \frac{s}{t} \quad \rightarrow \quad v_k = \frac{s_k - s_{k-1}}{t_k - t_{k-1}} \quad (k = 1, 2, 3, \ldots)$$

$$\Delta v = at; \quad s = \frac{at^2}{2} \quad \rightarrow \quad |s_k - s_{k-1}| = |\frac{v_k^2 - v_{k-1}^2}{2a}|$$

In the case study, the movement of the vehicle is based on the measured value of one IR sensor. Thereby, the measurement values of an IR sensor together with a confidence value are provided to the HiL simulator. The HiL simulator filters the distance value and calculates the position of the vehicle according to its physical model.

Process and Sensors

Besides implementing the process environment (i. e., the vehicle and its movement), the process itself as well as simulation models for the emulated sensors have to be considered. The process of the case study application is defined as follows:

- (1) Distance measurements are taken by three sensor nodes at nearly the same time.
- (2) Each sensor node calculates a value defining the confidence in the distance measurement.
- (3) All distance and confidence values are transmitted to a fusion node.
- (4) The fusion node processes the values, calculates an improved distance value and sends this value to a display node.
- (5) The distance value is displayed on a seven-segment display.

Implementation and Calibration

During HiL simulation, the fusion node of the case study executes either (1) a confidence weighted average algorithm or (2) a sensor selection algorithm for fusing the sensor measurements [Elm02]. Based on the experiments during the HiL simulation one of these fusion algorithms is to be selected for the final implementation.

The formula for the confidence weighted average algorithm is given by

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i \cdot \frac{1}{\mathbb{V}[S_i]}}{\sum_{i=1}^{n} \frac{1}{\mathbb{V}[S_i]}}$$

where n is the number of sensor measurements (three in our case), x_i represents the measured values, and $\mathbb{V}[S_i]$ is the estimated variance for measurement S_i. The values of the variance are derived from the confidence values of the sensor measurements.

Sensor selection means that the biggest and the smallest sensor measurement are eliminated and only the mean value is kept. Thereby, sensor selection adds fault tolerance to the fusion system.

HiL Simulation

The SUT of the case study includes the master node that implements the fusion algorithm (i. e., the master node and the fusion node are realized on the same hardware), one IR sensor node, and the display node. The HiL simulation is used to calibrate the sensor fusion algorithm and to pre-validate the application. The HiL simulation is based on a MATLAB/Simulink model executed on a low-cost desktop computer.

6.1.2 Elements

Figure 6.2 depicts the elements of the HiL simulation of the rear distance measurement system. The SUT consists of a combined master/fusion node, an IR sensor node, and a display node. The HiL simulator is realized by two components. First a TTP/A node acts as a combined *fieldbus FSC* and *gateway BSC* (refer to section 4.2).

Second, a desktop computer executes a MATLAB/Simulink model representing the environment of the SUT. The combined fieldbus FSC and gateway BSC node, or *gateway node* in short, is part of the HiL simulator.

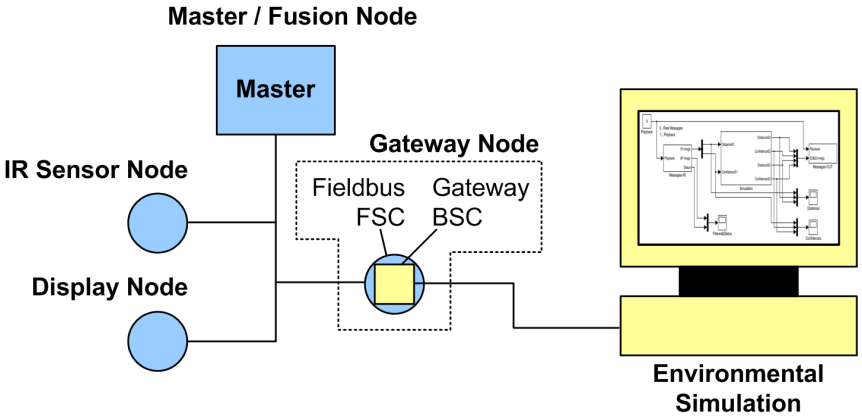

Figure 6.2: HiL simulation of rear distance measurement system

The purpose of the gateway node is to send simulated distance measurement values on the TTP/A bus at the same instants at which these values would be sent by physical IR sensor nodes. Hence, the gateway node implements the functionality of a fieldbus FSC. At the same time, the gateway node realizes the functionality of a gateway BSC, by providing an interface to the desktop computer executing the environmental simulation.

6.1.3 Implementation

As depicted in figure 6.3, the physical case study setup of the rear distance measurement system consists of a TTP/A cluster comprising a master(=fusion) node, an IR sensor node, a display node, a fieldbus FSC/gateway BSC node (gateway node), and the programmer. The programmer is used for downloading application code to the TTP/A nodes and is not further discussed in this thesis.

The gateway node is realized with an Atmel ATMega128 microchip. The gateway node is connected to the TTP/A cluster via its TTP/A interface and to the simulation host via its UART (RS232).

The filter application is executed on the TTP/A master node (an Atmel AT90S8515 microchip). The display is controlled by a TTP/A slave node (an Atmel AT90S4433 microchip). The IR smart transducer is based on an Atmel AT90S4433 microchip together with a Sharp GP2D12 IR sensor. The environmental simulation is executed on a Linux (kernel 2.6.2) based Intel i686 desktop computer.

The environmental simulation has been modeled with MATLAB/Simulink. Figure 6.4 shows the top-level view of the main simulation blocks, i. e., the *System Model* block and the *Measurement Model* block. The System Model block consists of two further blocks: *Weighted Average Dist* and *Vehicle Dynamics* (refer to figure 6.5).

At each simulation step, the first block which is called *Weighted Average Dist*

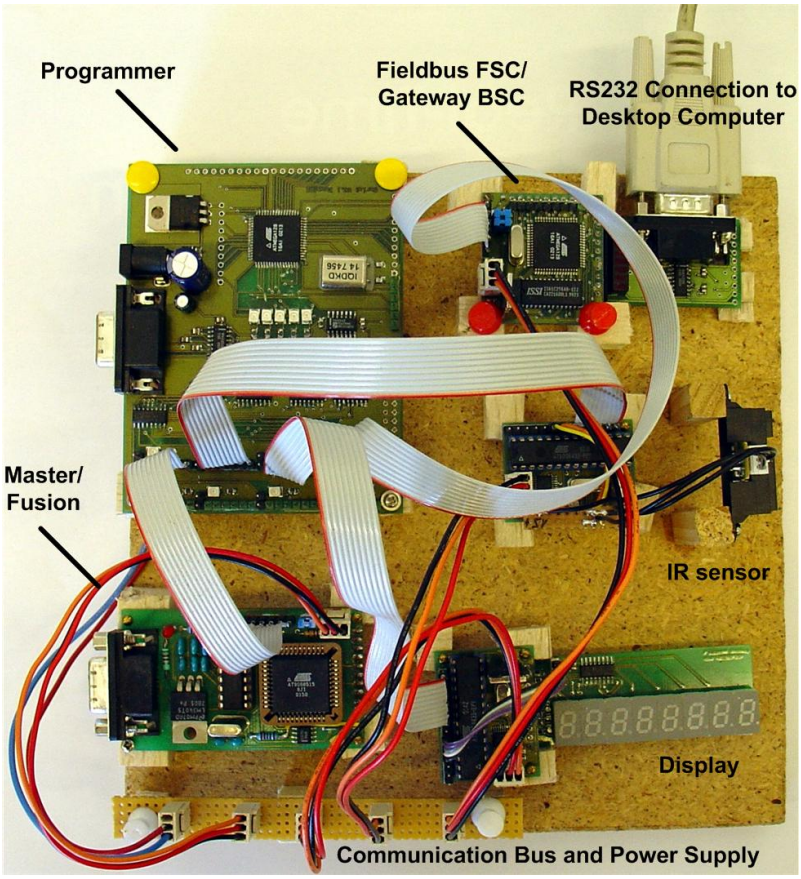

Figure 6.3: Physical setup of rear distance measurement system

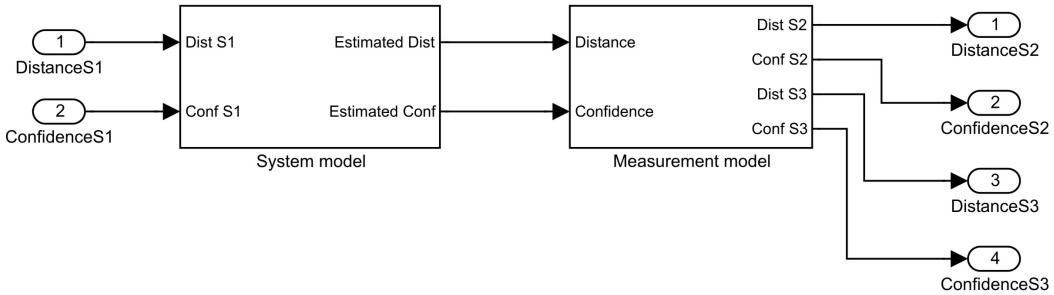

Figure 6.4: Simulation block

processes the five most recent sensor measurements of the physical IR sensor and calculates a weighted average in order to provide the second block with smoothed values. The second block called *Vehicle Dynamics* is provided with current (smoothed) sensor measurements. It calculates an approximation of the current position by combining on the one hand its knowledge of the physical characteristics of the car and on the other hand the sensor measurements of the physical IR sensor. Figure 6.6 depicts the internal view (implementation) of the Vehicle Dynamics block.

In the case study, the predicted (distance, confidence) pair of the system model

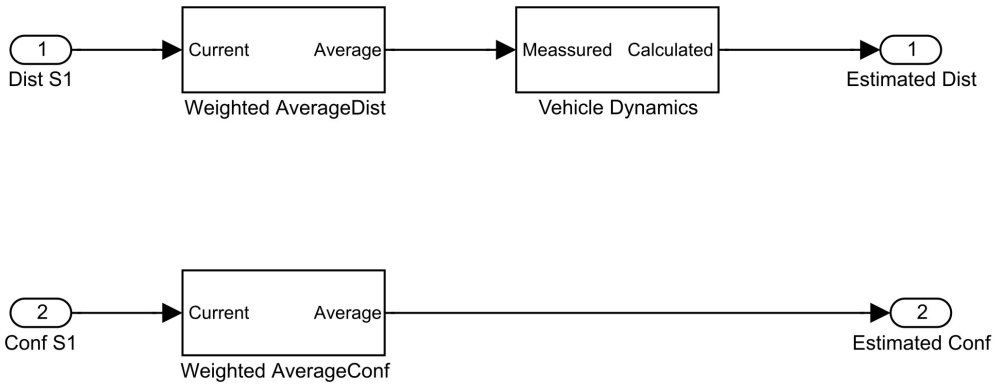

Figure 6.5: System model block

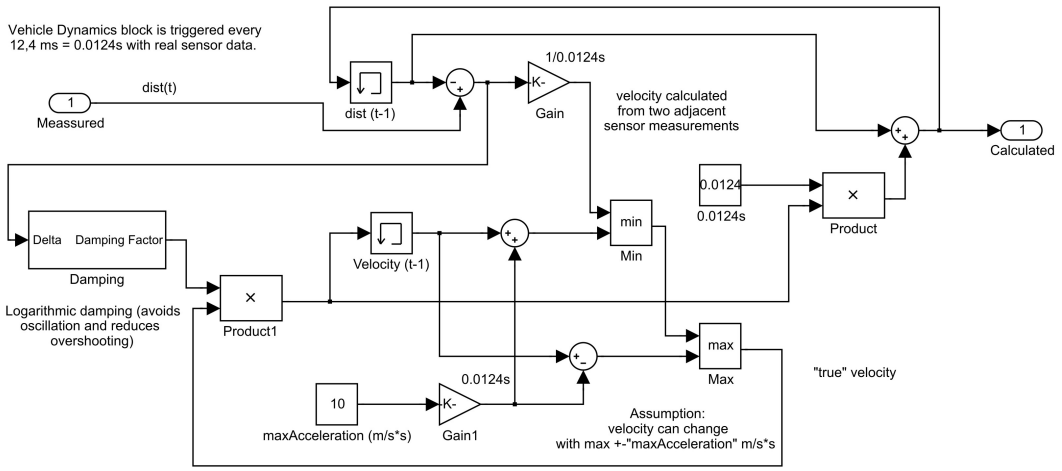

Figure 6.6: Vehicle dynamics block

block has been used as an input to the *Measurement Model* block. The Measurement Model block adds normally distributed random noise to the simulated IR sensor measurements.

Figure 6.7 depicts the internal view of a sensor measurement model. The precision of an IR sensor is assumed to be composed of two parts. The first part is a random disturbance value that does not depend on the current distance. The second part is a random disturbance value that depends on the distance value, i. e., this value increases when the distance value increases.

After realizing the simulation model and assembling the physical parts of the SUT, measurements have been taken to parameterize the implementation.

Figure 6.8 depicts a scenario, where the environment simulator processed the same input data with different values for maximum acceleration of the vehicle. The small line which shows fast changes of the distance values represents the actual measurements of the physically existing IR sensor. The other line shows the distance values of one simulated IR sensor which slowly follows the physical sensor (according to the vehicle model). The maximum acceleration for the vehicle model has been

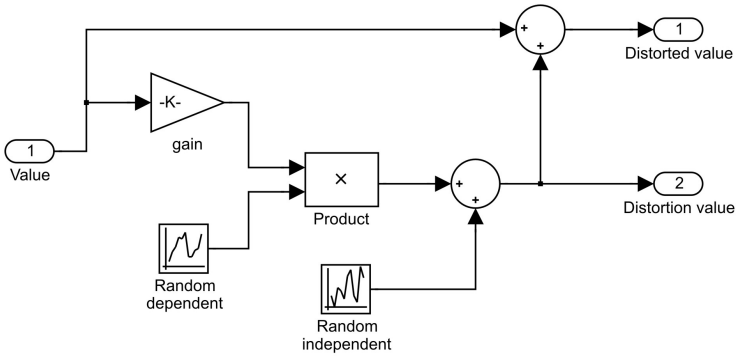

Figure 6.7: Distortion block

2 $\frac{m^2}{s}$, and 10 $\frac{m^2}{s}$ respectively. We can see that the thick line follows the smaller line faster when faster acceleration is allowed.

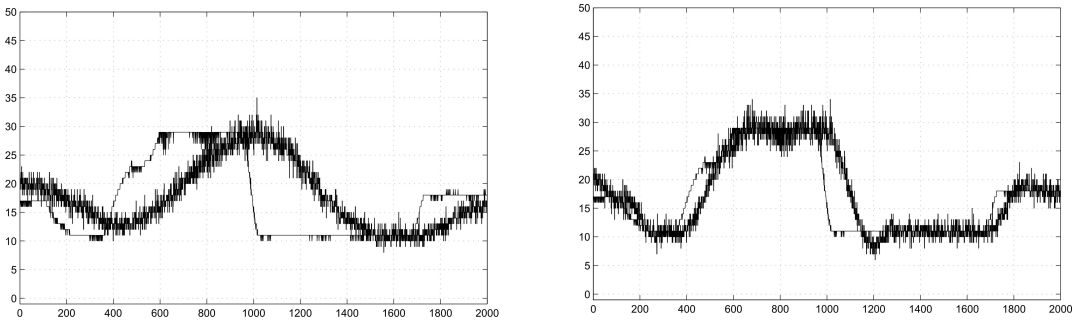

Figure 6.8: Simulated distances: max acceleration 2 $\frac{m^2}{s}$ and 10 $\frac{m^2}{s}$

Furthermore, several test runs have been performed in order to show the differences between dependent and independent disturbances of the measurements. Thereby, gaussian normally distributed random disturbances have been assumed. Figure 6.9 depicts two scenarios: In the left subpicture, the disturbance depends on the distance value (larger distance values imply greater disturbances), while in the right subpicture, the disturbance is constant (and does not depend on the distance value).

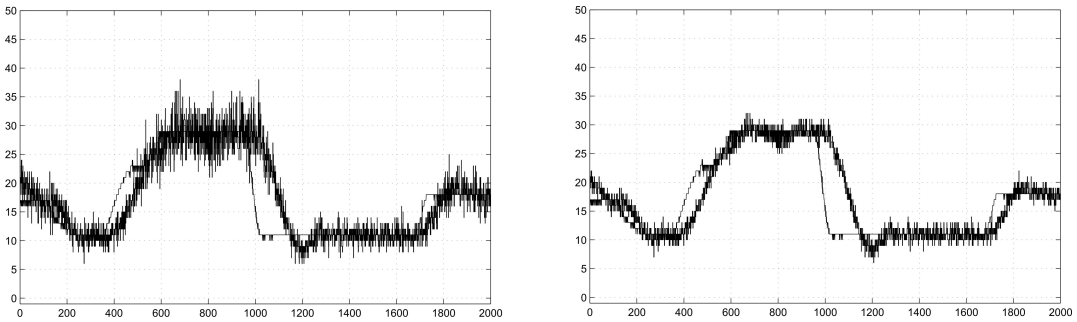

Figure 6.9: Simulated disturbances: dependent and independent

The results of the test-runs were compared by applying the same set of sensor

input data to the confidence weighted average algorithm and to the sensor selection algorithm. By matching the outputs (i. e., the fused distance measurements), it turned out that the confidence weighted average algorithm delivers better results – especially in the case of fast movement of the obstacle.

This is on the one hand due to better performance of the confidence weighted average algorithm in case of non-faulty sensors. On the other hand, the environmental simulation process does not allow fast changes of the simulated IR sensor measurements. Thus, the real IR sensor is marked faulty for a short period because both simulated IR sensors deliver values lying close together but being worse then the measurement of the real sensor.

6.1.4 Conclusion

The aim of the case study has been to exemplarily show the application of the HiL simulation framework to the calibration and testing of a rear distance measurement system.

The case study involves a soft real-time environmental simulation that is executed on a low-cost desktop computer and a gateway node that operates as a connection system between the environmental simulation and the SUT. The gateway node combines the functionality of a fieldbus FSC and a gateway BSC by providing a digital TTP/A transducer interface to the SUT and a second RS232 serial interface for connecting to the desktop computer.

Within the case study, the following features of the HiL simulation framework concept have experimentally been shown:

- The execution of the environmental simulation is done in soft real-time by a low-cost desktop computer. The gateway node requests simulation data from the environmental simulation and it is in the responsibility of the gateway node to timely interact with the SUT. Thus, temporal decoupling of the execution of the environmental simulation model from the timely interaction between the environmental simulation and the SUT has been realized.

- The model of the environmental simulation was developed with the graphical modeling and simulation tool MATLAB/Simulink. MATLAB/Simulink is a widely adopted tool that allows for easy development and modification of a simulation model. MATLAB/Simulink allows for model based (soft) real-time simulation and interaction with a physical target. Hence, graphical user interface blocks like the *Scope* block can be used during a simulation to visualize the characteristics of this simulation. Furthermore, the environment of MATLAB/Simulink enables monitoring and recording of simulation data which is captured by the gateway node. The features of MATLAB/Simulink have been used in the case study to implement the functionality of an HMI computer.

6.2 Control Path Simulation

6.2.1 Problem Statement

The idea behind the control path simulation case study is the development of a distributed control application for a heating installation [SWE06]. Within this case study, the heating element (i. e., the controlled object) is not available during development of the control application and must therefore be replaced by a simulation.

As depicted in figure 6.10, the distributed control application of the case study setup consists of a control node that contains a PI controller and two analog sensor interfaces, one for the actual value and one for the setpoint value. The first sensor interface is identical to the interface of an LM335Z temperature sensor from National Semiconductor. The other sensor interface is connected to a potentiometer that gives the setpoint value. The display node receives the actual value, the set value and the setpoint value from the network and displays these values at a 7-segment display. Another node acts as the TTP/A master. The actual target system would also have an actuator (heating element) with a TTP/A interface. In the case study setup this node is omitted and replaced by an HiL simulation.

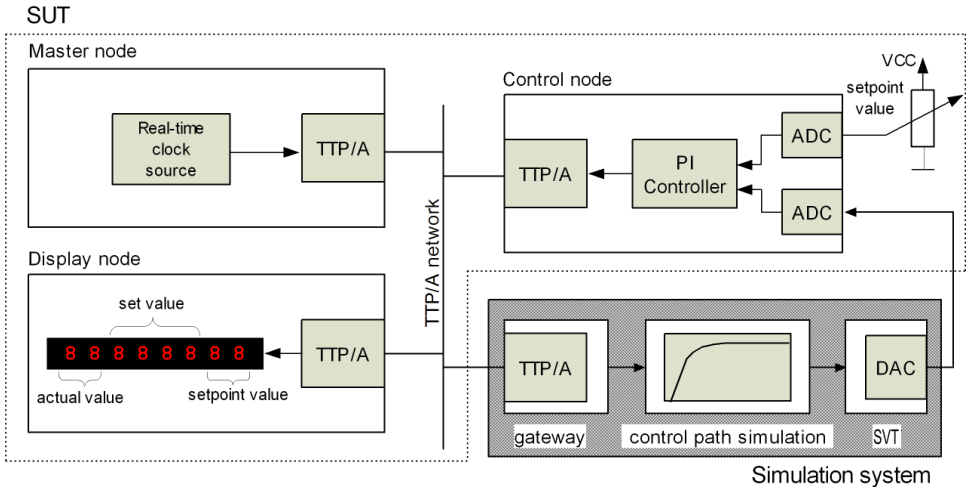

Figure 6.10: Control path simulation system

The TTP/A network is used to broadcast the actual value (8 bit), the setpoint value (8 bit) and the set value (16 bit) by the control node. The display node receives all three values, the gateway of the simulation requires only the set value. The communication bandwidth is 19.200 bit/sec; the cycle time of the cluster is 16,25ms.

The HiL simulator is part of the physical TTP/A network, i. e., it is implemented on a node with a TTP/A network interface, an Atmel AVR Atmega168 micro controller and an AD5330 digital/analog converter from Analog Devices. The purpose of the HiL simulator is to simulate a simple control path whereas the set value from the TTP/A bus acts as an input to the control path. The resulting value is converted into an analog signal which is forwarded to the control node. Within the case study

setup, an SVT node has been used for the HiL simulator implementation.

The case study involves the simulation of the heating element and the controller (including calibration of the controller).

Model of the Heating Element

We assume that the heating element is in accordance with a first order delay element (PT1 element), i. e., the heating element is represented through a proportional control path with a first order differential equation that gives the delay action. A characteristic of the control path for the heating element is a comparably slow response to changing inputs (set values).

The step response of the control path is given by:

$$ActualValue = SetValue * ks * (1 - e^{\frac{-t}{Ts}})$$

where ks stands for a constant describing the amplification of a specific heating element, t stands for time, and Ts for a time constant. In a discrete system as implemented within the case study, time is represented through discrete recuring instants i with period p. Hence, the equation given above can be transformed as follows:

$$ActualValue = SetValue * ks * (1 - e^{\frac{-i}{\frac{1}{p} * Ts}})$$

After selecting concrete values for amplification $ks = 1$, time constant $Ts = 1sec$, and period $p = 0.1sec$, the equation for the heating element is given as:

$$ActualValue = SetValue * 1 * (1 - e^{\frac{-i}{\frac{1}{0.1} * 1}})$$

which can be reduced to

$$ActualValue = SetValue * (1 - e^{\frac{-i}{10}}).$$

Model of the Controller

The controller providing the set value (u) of the heating element implements a PI controller. The algorithm of the PI controller is described through the following discrete formula:

$$u_n = u_{n-1} + kr * (e_n - e_{n-1} + \frac{10}{T_n + e_n})$$

with kr representing amplification, e_n standing for control deviation, and T_n giving the reset time.

6.2.2 Elements

As depicted in figure 6.11, the HiL simulation system of the control path case study involves three TTP/A nodes representing the SUT, namely a TTP/A master node, a control node executing the PI controller, and a display node that outputs the actual value, the set value and the setpoint value. Furthermore, the following three elements of the HiL simulator are part of the HiL simulation system:

Fieldbus FSC: The purpose of the fieldbus FSC is to capture the set value that is broadcasted on the TTP/A bus.

Basic BSC: The basic BSC provides the (PT1) control path simulation, i. e., it simulates the heating element.

Virtual Transducer FSC: The virtual transducer FSC generates an analog signal, thereby emulating the signal of an LM335Z temperature sensor.

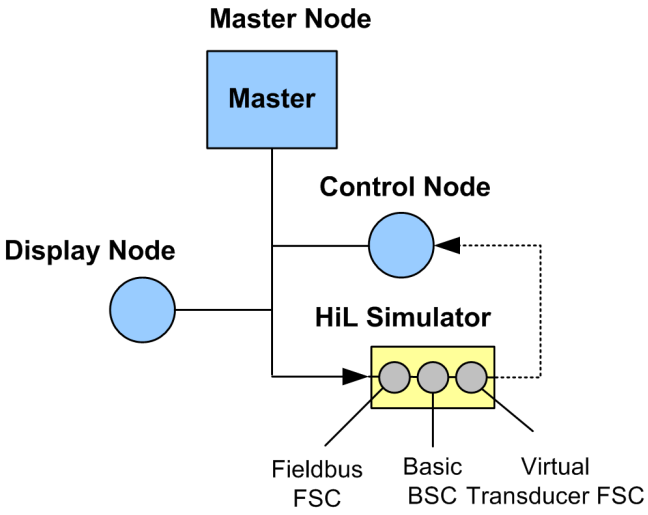

Figure 6.11: HiL simulation of control path system

The analog signal of the virtual transducer FSC is provided to the control node. Therewith, the loop between the PI controller and the PT1 control path simulation is closed.

6.2.3 Implementation

The implementation of the control path simulation case study is based on a TTP/A cluster design that defines the instants of sending/receiving messages and executing tasks. As outlined in section 2.4, the communication schedule of a TTP/A cluster is based on rounds that are subdivided into slots. It is possible to send a one-byte message in each of these slots. As shown in table 6.1, the master initiates a round by broadcasting the Fireworks Byte (FW). After that, each node either starts to execute (Ex), to send (Sd), or to receive (Rv) individual messages. The controller

	Slot 0	**Slot 1**	**Slot 2**	**Slot 3**	**Slot 4**
Master	FW				
Controller		Exe		Sd AV	Sd SPV
HiL Sim					
Display				Rv AV	Rv SPV
	Slot 5	**Slot 6**	**Slot 7**	**Slot 8**	**Slot 9**
Master					
Controller	Sd SV_l	Sd SV_h			
HiL Sim	Rv SV_l	Rv SV_h		Exe	
Display	Rv SV_l	Rv SV_h			Exe

Table 6.1: Cluster schedule

node provides the actual value (AV, 8bit, slot 3), the setpoint value (SPV, 8bit, slot 4), and the set value (SV, 16bit, slots 5 and 6).

In the following we investigate on the employed elements that have been used for the implementation of the case study.

Master Node

The master node has been implemented on an Atmel ATMega128, an 8bit RISC processor with 128 kBytes in-system flash, 4 kBytes EEPROM, and 4 kBytes SRAM. The ATMega128 includes 4 timers/counters, an USART, a programmable watchdog timer, Analog Digital Converters (ADCs), and an Serial Peripheral Interface (SPI) port.

The purpose of the master node is to initiate TTP/A rounds by sending the fireworks byte at the beginning of each round. Hence, the master node is regarded as the real-time clock source.

Controller Node

The PI controller node has been implemented on an Atmel ATMega128. The controller node receives the actual value and the setpoint value across its COI (i. e., via physical I/O ports), calculates the set value, and broadcasts this value.

Display Node

The display node has been implemented on an Atmel ATMega168, an 8bit RISC processor with 16 kBytes in-system flash, 512 bytes EEPROM, and 1 kByte SRAM. Furthermore, the ATMega168 offers 3 timers/counters, an USART, a 10bit ADC, a programmable watchdog, and an SPI port.

As depicted in figure 6.10, the display node is used for the purpose of visualizing the actual value, the setpoint value, and the set value. The display node controls 8 digits on 7-segment displays and updates the respective values according to the messages it receives from the controller node.

HiL Simulator Node

Within the case study, the elements of the HiL simulator have been implemented on a single SVT node that is based on an Atmel ATMega168 micro controller.

The HiL simulator node receives the set value from the controller node and calculates the actual value based on the set value (PT1 control path). The actual value is converted into an analog signal that corresponds to the analog signal of an LM335Z temperature sensor. This analog signal is provided to the controller node via a physical wire connecting the output of the HiL simulation node to the controller node.

A Velleman HPS5 oscilloscope has been used to visualize the step response of the control path. In figures 6.12 and 6.13, the step response has been captured which is based on a fast increase (from 0 to 128) of the setpoint value.

The step response given in figure 6.12 shows the control path when no controller is used, i. e., when the set value equals the setpoint value. Figure 6.13 depicts the step response of the control path when using a PI controller as discussed above.

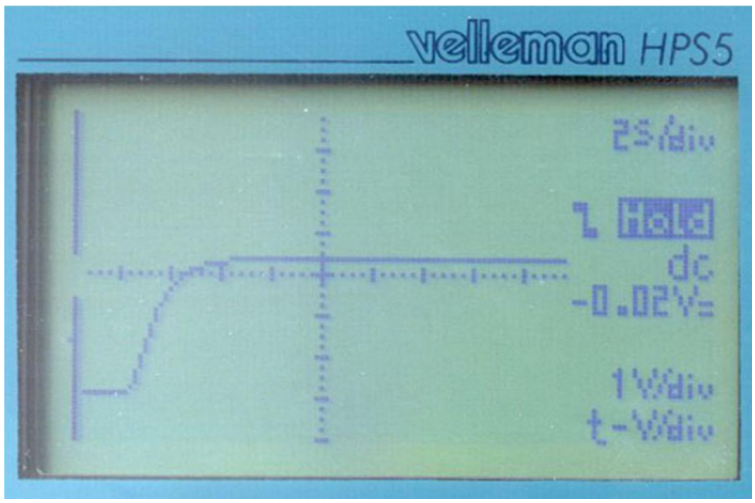

Figure 6.12: Step response without controller (set value = setpoint value)

6.2.4 Conclusion

The purpose of the control path simulation case study has been to demonstrate the capability of interaction between an SUT and an HiL simulator across an analog transducer interface (i. e., the interface of an LM335Z temperature sensor). In contrast to the previous case study (rear distance measurement), the simulation of the control path is executed on a simple BSC. This simple BSC receives its input values from a fieldbus FSC and provides its output values to a virtual transducer FSC.

The case study setup demonstrates the ability to calibrate a PI controller based on an HiL simulation setup that already employs the physical target hardware although the controlled object is not yet available.

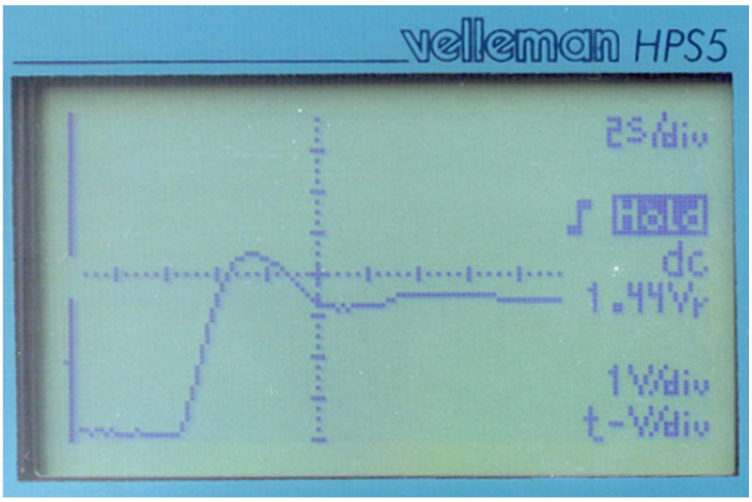

Figure 6.13: Step response with PI controller

Within the control path case study, the HiL simulator is fully integrated in the SUT. Hence, it is automatically synchronized to the real-time clock source provided by the master node. Nevertheless, the HiL simulation does not introduce any probe effect on the SUT. The HiL simulator node passively listens to the transmitted messages but does not send any message by itself.

6.3 Rear Parking System

6.3.1 Problem Statement

The idea of the rear parking system case study is to set up an HiL simulation of a driver assistance system (similar to the digital smart transducer gateway case study). The driver assistance system consists of infra-red and ultra-sonic distance measurement sensors as well as a display and a haptic steering wheel indicating approaching obstacles when backing the car.

The rear parking system case study augments the digital smart transducer gateway case study by involving time-dependent sensors (ultra sonic sensors). Furthermore, the interaction between a distributed HiL simulator and the SUT is realized through SVTs (i. e., virtual transducer FSCs).

The case study includes the simulation of a Polaroid 6500 series sonar ranging transducer [Wir97] by an SVT (ultrasonic SVT). A Polaroid 6500 ultra-sonic transducer can be instrumented to operate in single-echo mode. In this mode of operation the INIT input of the transducer is set to high in order to start the transmission of an acoustic signal (16 pulses at 49.4 kHz with 400 volt amplitude). As soon as the echo of this acoustic signal has been received back, the ECHO output of the transducer is set to high. The interval between INIT high and ECHO high is proportional to the distance to the measured object (refer to figure 6.14).

Each ultrasonic SVT is periodically provided with the actual distance value from

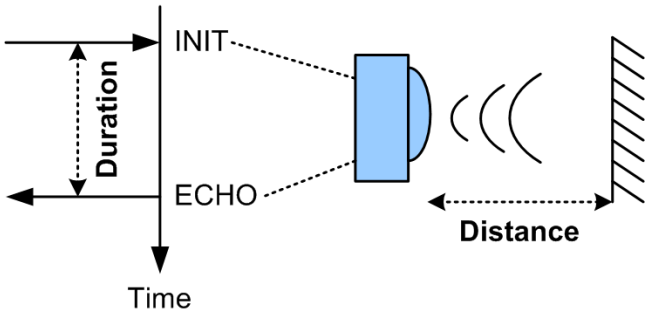

Figure 6.14: Distance measurement with an ultra-sonic sensor

a BSC. Regarding the physical interconnection to the SUT, an SVT offers an INIT and an ECHO port (digital I/O of the SVT). The ultrasonic SVT polls the INIT input with high frequency. As soon as the input is set to high, a timer is set in accordance with the current distance value. This timer is used to set the ECHO signal of the SVT to high after a specified amount of time.

6.3.2 Elements

Figure 6.15 depicts the elements of the rear parking system case study. The SUT consists of a master node (Master), a node that controls the motor of the haptic steering wheel (Motor), three nodes that are connected to infra-red distance sensors (IR0, IR1, IR2), two nodes that are connected to ultra-sonic distance sensors (US0, US1), and a node that controlls a display (Display).

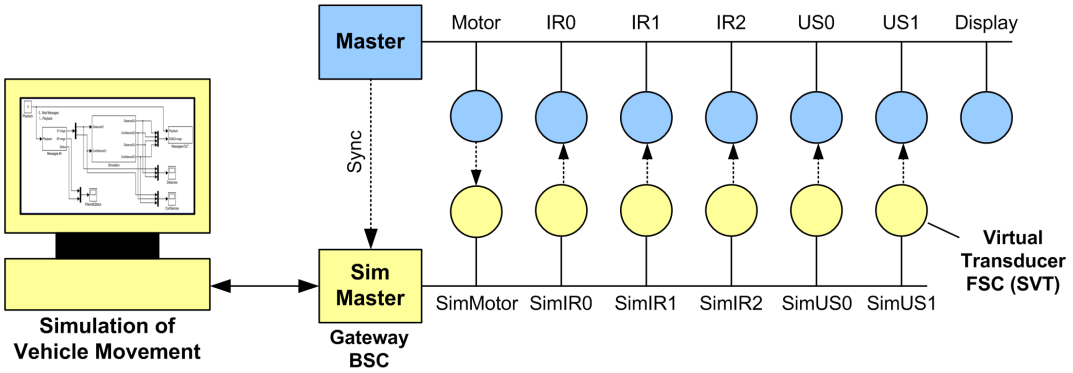

Figure 6.15: HiL simulation of rear parking system

The HiL simulator comprises six SVTs that simulate the motor, the infra-red sensors, and the ultra-sonic sensors, respectively. Furthermore, the master node (Sim-Master) acts as a gateway BSC that connects the SVTs to a desktop computer. This desktop computer executes the simulation of the environment of the SUT, i. e., the movement of the vehicle towards an obstacle.

6.3.3 Implementation

Figure 6.16 shows a picture of the physical setup of the rear parking system case study with two separate TTP/A clusters (i. e., SUT cluster and HiL simulator cluster).

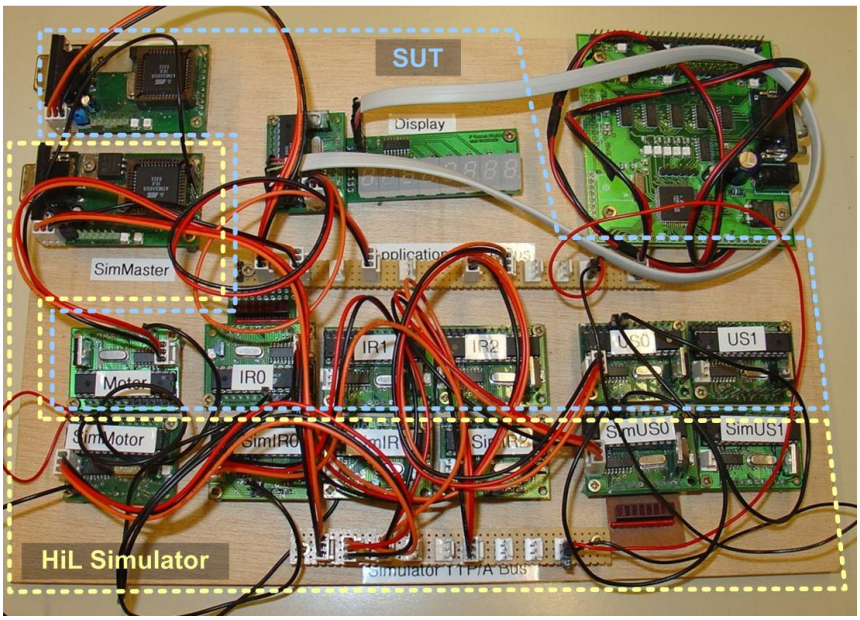

Figure 6.16: Physical setup of rear parking system

The SUT consists of a TTP/A master node, five sensor nodes (IR0, IR1, IR2, US0, US1) and two actuator/display nodes (Motor, Display). These nodes are interconnected via TTP/A. The sensor nodes can interface both digital sensors (digital input ports) and analog sensors (built in A/D converter). The sensor nodes capture distance measurements in the range of 10cm to 200cm and send these measurements as TTP/A messages to the master node. The master node fuses the individual measurements and sends messages to the motor node and to the display node.

The HiL simulator consists of six SVTs, each of them simulating a dedicated sensor/actuator (SimMotor, SimIR0, . . .). The SVTs that simulate infra-red sensors are connected to the IR sensor nodes of the SUT via an analog interface. The measured distance of an IR sensor is represented through the voltage level of the IR sensor. The SVTs that simulate ultra-sonic sensors are connected to the ultra-sonic sensor nodes of the SUT via a digital interface (as described above, the ultra-sonic sensor is initiated and sends back a signal after an interval whose length depends on the measured distance of the sensor). The SVT that simulates the motor accepts a digital (PWM) signal from the motor control node of the SUT.

The master node of the HiL simulator additionally acts as a gateway BSC establishing a connecting to a desktop computer via the serial (RS232) interface (in figure 6.16, the master node is not connected to the desktop computer).

6.3.4 Conclusion

In contrast to the previous case studies, the rear parking system case study involves a time-dependent ultra sonic sensor. Furthermore, the interaction between the HiL simulator and the SUT is realized through SVTs (i. e., virtual transducer FSCs), including both, analog and digital physical interfaces.

6.4 Chapter Summary

In this chapter three different case studies have been presented, each of them focusing on a particular aspect of the previously discussed HiL simulation concepts. For each of these case studies, the respective problem statement, major elements, and implementation have been discussed.

The digital smart transducer gateway case study involves the prototype of a rear distance measurement system in a car. The process model for this system has been formalized and the process as well as the physical transducers have been modeled. After that, the system (i. e., the SUT) has been implemented and parameterized and HiL tests have been applied. The HiL simulator of this case study includes a combined fieldbus FSC and gateway BSC which connects the SUT to a desktop computer. The desktop computer is part of the HiL simulation and executes a MATLAB/Simulink model representing the environment of the SUT.

The control path simulation case study implements a distributed control application for a heating installation. This application consists of a control node containing a PI controller and two analog sensor interfaces for the actual value and the setpoint value, respectively. The HiL simulator of this case study has been implemented on a single hardware node representing a fieldbus FSC, a basic BSC, and a virtual transducer FSC. The control path simulation case study demonstrates the ability of interfacing an SUT across an analog transducer interface.

The rear parking system case study extends the digital smart transducer gateway case study by (1) adding time-dependent (ultra-sonic) distance measurement sensors and (2) realizing the interaction between the SUT and the HiL simulator through SVTs. The HiL simulator of this case study includes SVTs for the simulation of a motor, ultra-sonic sensors, and infra-red sensors. Furthermore, the HiL simulator involves a gateway BSC connecting the SVTs to a desktop computer.

Chapter 7

Conclusion

The thesis provides a systematic approach for the predictable real-time interaction between an Hardware-in-the-Loop (HiL) simulator and a System-Under-Test (SUT). With such an approach, it can be guaranteed that the information flow from the environmental simulation of the HiL simulator to the SUT (and vice versa) is bound to *a priori* known latency and jitter. The prototypically implemented concept of a Smart Virtual Transducer (SVT) facilitates the interaction between an HiL simulator and an SUT across a wide range of different transducer interfaces. The concept can be exploited in the future for modular, cost-efficient HiL simulator implementations.

7.1 HiL Simulation Framework

Within this thesis, a scalable, component-based, deterministic HiL simulation framework has been presented. Based on this framework, it is possible to construct a distributed HiL simulator combining the functionality of an environment simulator and (optionally) a cluster simulator.

The proposed HiL simulation framework involves a set of Backend Simulation Components (BSCs) and Frontend Simulation Components (FSCs). An FSC controls the physical interconnection between the HiL simulator and the SUT, thereby consisting of one digital interface to a TTP/A network and another interface resembling the Controlled Object Interface (COI) of the SUT. The second interface is either a digital fieldbus interface, an arbitrary transducer interface, or a physical transducer. A BSC is, in terms of physical interconnection between the HiL simulator and the SUT, independent of low level details and is used to execute part of a distributed simulation model. It has a digital interface to the TTP/A network and optionally a second digital gateway interface connecting to an external system.

An essential requirement of the HiL simulation framework has been to establish predictable and precise instants with respect to the interaction between the HiL simulator and the SUT. This interaction depends on the progression of a synchronized sparse (real-)time and is described through well-specified interfaces (both in the temporal and functional domain).

A distributed HiL simulator, assembled according to the presented framework, allows for parallel development, scalability, complexity management, and multirate simulation. Apart from supporting arbitrary transducer interfaces at the COI of the SUT, the HiL simulator can be implemented to provide mechanisms for monitoring, run-time configuration, and data logging.

7.2 Smart Virtual Transducer

Within the thesis, the new concept of an SVT and the prototype implementation of an SVT component have been shown. An SVT is used as a connection system to interconnecting an HiL simulator and an SUT across an arbitrary transducer interface. Thereby, an SVT mimics the behavior of a physical transducer (i. e., sensor or actuator) during an HiL simulation run. An SVT consists of a processing unit, a communication interface, and a transducer-specific interface. Contrary to a Smart Transducer (ST), an SVT does not contain a physical sensor or actuator element.

An SVT offers two interfaces, the Object Management Group (OMG) Smart Transducer Interface (STI) network interface and a virtual transducer interface. The interaction between the SVT and the SUT takes place via the virtual transducer interface. A sensor SVT emulates a sensor, an actuator SVT emulates an actuator. This emulation can be done in the value (e. g., infrared sensor, Light Emitting Diode (LED)) or in the time domain (e. g., ultrasonic sensor, Pulse-Width Modulation (PWM) controlled motor).

The concept of SVT leads to a reduction of the cognitive complexity when setting up an HiL simulation system. In particular, the simulation model (which is executed at the BSCs) is decoupled from the SVT elements interacting with the SUT (*temporal laxity*). The low-level behavior of a particular transducer element is covered using the parts of the simulation model, which are executed at the BSCs. Potential changes of the low-level properties of transducer elements (e. g., upgrade of a transducer to a newer model) do not directly influence the BSC because simulation models of the transducers are confined by the SVTs.

Furthermore, the SVT approach supports reusability by the provision of configurable FSCs. Since the implementation of an SVT mainly depends on the transducer that is replaced, an SVT can be reused in other applications whenever the same kind of transducer is employed. Features such as frequency of update values and smoothing parameters depend on the control environment. However, for a given application these functions can be implemented and parametrized generically.

7.3 Outlook

The presented HiL simulation framework builds upon well established concepts of the Time-Triggered Architecture (TTA) and makes use of the time-triggered fieldbus protocol TTP/A in order to deterministically interconnect a network of simulation components. Based on these concepts, most prerequisites for the exploitation of the

HiL simulation framework are already fulfilled, such as the existence of a communication infrastructure for predictable message transmission, a sparse global timebase, or the concept of a temporal firewall along with precisely defined interfaces.

Nevertheless, in the area of the HiL simulation framework, several future advancements are conceivable. Such advancements should built on the exemplarily outlined application of the HiL simulation framework for the validation of (large) integrated real-time systems in safety-related fields (e. g., aerospace, automotive, medical, or industrial control) and could be categorized into (a) future work regarding the implementation of simulation framework elements and (b) future work with respect to simulation tools guiding a developer through design and deployment phases.

The realization of simulation framework elements involves the implementation of a library of generic SVT components that can be readily configured for specific transducer elements. Furthermore, the utilization of a time-triggered protocol with higher bandwidth (e. g., Time-Triggered Protocol (TTP), FlexRay) can be used to allow faster interaction between components of the HiL simulator (thereby allowing faster simulation steps). Future work on simulation tools will lead to a comprehensive tool-chain towards the development of an HiL simulation combining tools for designing and executing simulation models (including Worst Case Execution Time (WCET) analysis), configuring the HiL simulator, and monitoring/managing simulation outputs.

List of Acronyms

ADC Analog Digital Converter

AUTOSAR AUTomotive Open System ARchitecture

BSC Backend Simulation Component

CAN Controller Area Network

CHI Controller Host Interface

CNI Communication Network Interface

COI Controlled Object Interface

CORBA Common Object Request Broker Architecture

COTS Commercial-Off-The-Shelf

CP Configuration and Planning

DAC Digital Analog Converter

DAS Distributed Application Subsystem

DECOS Dependable Embedded COmponents and Systems

DSP Digital Signal Processing

DM Diagnostic and Management

DSoS Dependable Systems of Systems

ECU Electronic Control Unit

FAA Federal Aviation Administration

FPGA Field Programmable Gate Array

FSC Frontend Simulation Component

GWI Gateway Interface

HiL Hardware-in-the-Loop

HMI Human Machine Interface

IEC International Electrotechnical Commission

IFS Interface File System

IMA Integrated Modular Avionics

ISA Instrumentation Society of America

ISUT Integrated System-Under-Test

JTAG Joint Test Action Group

LED Light Emitting Diode

LIF Linking Interface

LIN Local Interconnect Network

LLI Logical Line Interface

MEDL Message Descriptor List

MiL Model-in-the-Loop

MISRA Motor Industry Software Reliability Association

OMG Object Management Group

PCI Peripheral Component Interconnect

PiL Processor-in-the-Loop

PROFIBUS Process Field Bus

PWM Pulse-Width Modulation

PXI PCI eXtensions for Instrumentation

RODL Round Descriptor List

RS Real-Time Service

RTCA Radio Technical Commission for Aeronautics

SiL Software-in-the-Loop

SOPC System-On-a-Programmable-Chip

SPI Serial Peripheral Interface

SPLIF Service Providing Linking Interface

SRLIF Service Requesting Linking Interface

SSI Simulation System Interface

ST Smart Transducer

STI Smart Transducer Interface

SUT System-Under-Test

SVT Smart Virtual Transducer

TAI Temps Atomique Internationale

TCK Test ClocK

TDI Test Data Input

TDMA Time Division Multiple Access

TDO Test Data Output

TTA Time-Triggered Architecture

TTBN Time-Triggered Backbone Network

TTP Time-Triggered Protocol

UART Universal Asynchronous Receiver Transmitter

UTC Universal Time Coordinated

WCET Worst Case Execution Time

Bibliography

[AD00] Analog Devices. *Data Sheet of AD5330/AD5331/AD5340/AD5341*, 2000.

[ADI05] Applied Dynamics International. *Distributed HIL Simulation*, 2005. Available at http://www.adi.com.

[AFI+00] J. Arlat, J.C. Fabre, V. Issarny, M. Kaâniche, K. Kanoun, C. Kloukinas, B. Marre, E. Marsden, D. Powel, A. Romanovsky, P. Thévenod-Fosse, H. Waeselynck, I. Welch, I. Zakkiudin, and A. Zarras. Dependable systems of systems (DSoS), State of the art survey, Deliverable BC2. Research Report 00353, University of Newcastle upon Tyne, April 2000. Available at http://rogue.ncl.ac.uk.

[AHC05] M. Andersson, D. Henriksson, and A. Cervin. *TrueTime 1.3—Reference Manual*, June 2005. Available at http://www.control.lth.se.

[AHCÅ05] M. Andersson, D. Henriksson, A. Cervin, and K.E. Årzén. Simulation of wireless networked control systems. In *Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference (ECC'05)*, Seville, Spain, December 2005.

[ALRL04] A. Avižienis, J.C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, January–March 2004.

[ARI91] Aeronautical Radio Incorporated (ARINC), Annapolis, MD, USA. *ARINC Specification 651: Design Guide for Integrated Modular Avionics*, November 1991.

[ASBT03] A. Ademaj, H. Sivencrona, G. Bauer, and J. Torin. Evaluation of fault handling of the time-triggered architecture with bus and star topology. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 123–132, 2003.

[ATI01] ATIS Committee T1A1, American National Standards Institute, Inc. Telecom glossary 2000, February 2001.

[Atm04] Atmel Corporation. Data sheet of ATmega48, ATmega88, and ATmega168, 2004.

[AUT06a] AUTOSAR GbR. *AUTOSAR – Glossary V2.0.1*, June 2006.

[AUT06b] AUTOSAR GbR. *AUTOSAR – Technical Overview V2.0.1*, June 2006.

[BAD05] J. Bélanger, S. Abourida, and C. Dufour. Real-time digital simulation and control laboratory for distributed power electronic generation and distribution. In *Proceedings of the Huntsville Simulation Conference (HSC'05)*, Huntsville, AL, USA, October 2005.

[Bal97] O. Balci. Verification, validation and accreditation of simulation models. In *Proceedings of the 1997 Winter Simulation Conference*, pages 135–141, Atlanta, GA, USA, December 1997.

[BB98] P. Bishop and R. Bloomfield. A methodology for safety case development. In *Proceedings of the Sixth Safety-critical Systems Symposium on Industrial Perspectives of Safety-critical Systems*, Birmingham, UK, 1998.

[BCNN01] J. Banks, J.S. Carson, B.L. Nelson, and D.M. Nicol. *Discrete-Event System Simulation*. Prentice Hall, Upper Saddle River, NJ, USA, 2001.

[BE05] W. Bernhart and H.P. Erl. Markt- und Technologiestudie Leistungselektronik Automotive, A.D.Little, September 2005.

[Bro05] M. Broy. Automotive software and systems engineering. In *Proceedings of the 3rd ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE'05)*, Verona, Italy, July 2005.

[BSS05] T. Bäro, E. Sax, and S. Schmerler. Erhöhung der Testtiefe durch HiL-Testing. In *Proceedings der Jahrestagung der ASIM/GI-Fachgruppe 4.5.5 Simulation technischer Systeme*, pages 4–13, Berlin, Germany, March 2005.

[BV99] I.J. Busch-Vishniac. *Electromechanical Sensors and Actuators*. Springer, 1999.

[CDK05] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design (4th Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[Clo98] D.J. Cloud. *Applied Modeling and Simulation*. McGraw-Hill Primis Custom Pub., 1998.

[Cra05] R. Cravotta. Mixing the real with the virtual. *EDN*, pages 57–62, May 2005.

[DSP06] Altera Corporation. *DSP Builder – User Guide*, April 2006.

[Ecc00] D.S. Eccles. Building simulators for aerospace applications: processes, techniques, choices and pitfalls. In *Aerospace Conference Proceedings, IEEE*, pages 517–527. IEEE, March 2000.

[EHK01] W. Elmenreich, W. Haidinger, and H. Kopetz. Interface design for smart transducers. In *IEEE Instrumentation and Measurement Technology Conference (IMTC)*, volume 3, pages 1642–1647, May 2001.

[EK05] W. Elmenreich and S.V. Krywult. A comparison of fieldbus protocols: LIN 1.3, LIN 2.0, and TTP/A. In *10th IEEE International Conference on Emerging Technologies and Factory Automation*, Catania, Italy, September 2005.

[EKRZ04] D. Edenfeld, A.B. Kahng, M. Rodgers, and Y. Zorian. 2003 technology roadmap for semiconductors. *Computer, IEEE Computer Society*, 37(1):47–56, January 2004.

[Ell00] M. Elliott. Simulation buyer's guide. Technical report, Institute of Industrial Engineers (IIE), 25 Technology Park, Norcross, GA 30092, USA, May 2000.

[Elm02] W. Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 3/3/182-1, 1040 Vienna, Austria, 2002.

[EO02] W. Elmenreich and R. Obermaisser. A standardized smart transducer interface. In *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE'02)*, July 2002.

[EP03] W. Elmenreich and S. Pitzek. Smart transducers – principles, communications, and configuration. In *Proceedings of the 7th IEEE International Conference on Intelligent Engineering Systems*, volume 2, pages 510–515, Assuit – Luxor, Egypt, March 2003.

[EPS04] W. Elmenreich, S. Pitzek, and M. Schlager. Modeling distributed embedded applications on an interface file system. In *Proceedings of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 175–182, May 2004.

[ES04] W. Elmenreich and M. Schlager. Simulation-based development of embedded sensor fusion applications. In *Proceedings of the 2nd IEEE International Conference on Computational Cybernetics*, pages 147–153, September 2004.

[FBMD06] H.P. Figueroa, J.L. Bastos, A. Monti, and R. Dougal. A modular real-time simulation platform based on the virtual test bed. In *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE'06)*, pages 1537–1541, Montréal, Canada, July 2006.

[Fel02] M. Felser. The fieldbus standard: History and structure. In *Technology Leadership Day 2002*. MICROSWISS Network, HTA Luzern, October 2002.

[FRB97] W. Fleisch, T. Ringler, and R. Belschner. Simulation of application software for a TTP real-time subsystem. In *Europaen Simulation Multiconference (ESM)*, Istanbul, Turkey, June 1997.

[Fre05] Freescale Semiconductor. *MFR4200 Datasheet FlexRay Communication Controllers*, August 2005.

[Fro04] Frost & Sullivan, 2400 Geng Road, Suite 201, Palo Alto, CA 94303-3331, US. *Strategic Analysis of European Semiconductor Market for Passenger Cars*, April 2004.

[FX05] FlexRay Consortium. BMW AG, DaimlerChrysler AG, General Motors Corporation, Freescale GmbH, Philips GmbH, Robert Bosch GmbH, and Volkswagen AG. *FlexRay Requirements Specification Version 2.1*, December 2005.

[Gai86] J. Gait. A probe effect in concurrent programs. *Software Practice and Experience*, 16(3):225–233, March 1986.

[Gal99] T. Galla. *Cluster Simulation in Time-Triggered Real-Time Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 3/3/182-1, 1040 Vienna, Austria, 1999.

[GFL+02] P. Giusto, A. Ferrari, L. Lavagno, J.-Y. Brunel, E. Fourgeau, and A. Sangiovanni-Vincentelli. Automotive virtual integration platforms: why's, what's, and how's. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 370–378, September 2002.

[GLH06] S. Gilbert and H. Le-Huy. A comparative study on real-time simulation methods for PWM power converters. In *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE'06)*, pages 2571–2578, Montréal, Canada, July 2006.

[Gom01] M. Gomez. Hardware-in-the-loop simulation, November 2001. Embedded Systems Programming, CMP Media LLC, 600 Harrison Street, San Franzisco, CA, 94107.

[Gri01] P. Grillinger. Simulation model of TTP/C protocol. In *Proceedings of the 23rd International Autumn Colloquium – Advanced Simulation of Systems (ASIS'01)*, Velke Losiny, Czech republic, September 2001.

[Han04] A. Hanzlik. *Investigation of Fault-Tolerant Multi-Cluster Clock Synchronization Strategies by Means of Simulation*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 3/3/182-1, 1040 Vienna, Austria, 2004.

[HCÅ03] D. Henriksson, A. Cervin, and K.E. Årzén. TrueTime: Real-time control system simulation with MATLAB/Simulink. In *Proceedings of the Nordic MATLAB Conference*, Copenhagen, Denmark, October 2003.

[HHBC06] W. Herzner, B. Huber, A. Balogh, and G. Csertan. The DECOS toolchain: Model-based development of distributed embedded safety-critical real-time systems. *DECOS/ERCIM Workshop on Dependable Embedded Systems at SAFECOMP, Gdansk, Poland*, September 2006.

[Hoe03] C. Hoefer. "Causal Determinism" — Stanford Encyclopedia of Philosophy, 2003.

[Hoe04] C. Hoefer. Causality and determinism: Tension, or outright conflict? *Revista de Filosofía*, 29(2):99–115, November 2004.

[HPOES05] B. Huber, P. Peti, R. Obermaisser, and C. El-Salloum. Using RTAI/-LXRT for partitioning in a prototype implementation of the DECOS architecture. In *Proceedings of the Third International Workshop on Intelligent Solutions in Embedded Systems*, May 2005.

[IEC05] International Electrotechnical Commission (IEC) SC65A/WG14, IEC, Geneva, Switzerland. *Functional safety and IEC 61508*, September 2005.

[IEE89] IEEE Standards Board. IEEE standard glossary of modeling and simulation terminology. Technical Report IEEE Std 610.3-1989, The Institute of Electrical and Electronics Engineers, Inc 345 East 47th Street, New York, NY 10017, USA, 1989.

[IEE90] IEEE Standards Board. IEEE standard glossary of software engineering terminology. Technical Report IEEE Std 610.12-1990, The Institute of Electrical and Electronics Engineers, Inc 345 East 47th Street, New York, NY 10017, USA, 1990.

[IMA05] Radio Technical Commission for Aeronautics (RTCA) Inc., WG60/SC200 Plenary. *Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations (Rev M+)*, April 2005.

[Jim05] H.P.F. Jimenez. *Novel Interface Based on the Gating Signal Averaging Method for Accurate Hardware-In-The-Loop Testing of Digital Controllers for Power Electronics Applications*. PhD thesis, University of South Carolina, College of Engineering and Information Technology, Department of Electrical Engineering, Swearingen Engineering Center, 301 South Main Street, Columbia, SC 29208, USA, 2005.

[JKK+02] C. Jones, M.-O. Killijian, H. Kopetz, E. Marsden, N. Moffat, D. Powell, B. Randell, A. Romanovsky, R. Stroud, and V. Issarny. Final version of the DSoS conceptual model. Research Report 54/2002, DSoS Project (IST-1999-11585), October 2002. Deliverable CSDA1, available at http://www.vmars.tuwien.ac.at.

[KB03] H. Kopetz and G. Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 91(1):112 – 126, January 2003.

[KEM00] H. Kopetz, W. Elmenreich, and C. Mack. A comparison of LIN and TTP/A. In *3rd IEEE International Workshop on Factory Communication Systems (WFCS'00)*, pages 99–107, Porto, Portugal, September 2000.

[KESHO07] H. Kopetz, C. El-Salloum, B. Huber, and R. Obermaisser. Periodic finite-state machines. In *Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, Santorini Island, Greece, May 2007.

[KFMN99] H. Kopetz, E. Fuchs, D. Millinger, and R. Nossal. An interface as a design object. *2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'99), 2-5 May 1999*, May 1999.

[KHE01] H. Kopetz, M. Holzmann, and W. Elmenreich. A universal smart transducer interface: TTP/A. *International Journal of Computer System Science & Engineering*, 16(2):71–77, March 2001.

[Kil01] R.A. Kilgore. Open source simulation modeling language (SML). In *Proceedings of the 2001 Winter Simulation Conference*, pages 607–613, Washington, DC, USA, 2001.

[Kim04] K.H. Kim. The distributed time-triggered simulation scheme: Core principles and supporting execution engine. *Real-Time Systems*, 26(1):9–28, 2004.

[KN97] H. Kopetz and R. Nossal. Temporal firewalls in large distributed real-time systems. In *Proceedings of the 6th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 310–315, Tunis, Tunesia, October 1997.

[Kop92] H. Kopetz. Sparse time versus dense time in distributed real-time systems. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, Yokohama, Japan, June 1992.

[Kop93] H. Kopetz. Should responsive systems be event-triggered or time-triggered? *Institute of Electronics, Information, and Communications Engineers Transactions on Information and Systems*, E76-D(11):1325–1332, 1993.

[Kop97] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[Kop99a] H. Kopetz. Do current technology trends enforce a paradigm shift in the industrial automation market? *7th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'99), at Barcelona, Spain, 18-22 October 1999*, October 1999.

[Kop99b] H. Kopetz. Elementary versus composite interfaces in distributed real-time systems. *ISADS'99, March 1999, Tokyo, Japan*, March 1999.

[Kop02] H. Kopetz. Time-triggered real-time computing. *IFAC World Congress, IFAC Press*, July 2002.

[Kop04] H. Kopetz. On the fault hypothesis for a safety-critical real-time system. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS'04)*, Florianpolis, Brazil, October 2004.

[KOPS04] H. Kopetz, R. Obermaisser, P. Peti, and N. Suri. From a federated to an integrated architecture for dependable embedded real-time systems. Technical Report 22/2004, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 1-3/182-1, 1040 Vienna, Austria, 2004.

[Krü97] A. Krüger. *Interface Design for Time-Triggered Real-Time System Architectures*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 3/3/182-1, 1040 Vienna, Austria, 1997.

[KS03] H. Kopetz and N. Suri. Compositional Design of RT Systems: A Conceptual Basis for Specification of Linking Interfaces. *6th IEEE International Symposium on Object-Oriented Real-Time Computing (ISORC'03)*, May 2003.

[Lap92] J.C. Laprie. *Dependability: Basic Concepts and Terminology*. Springer-Verlag, 1992.

[Lap04] P.A. Laplante. *Real-Time Systems Design and Analysis*. IEEE Press, 2004.

[Law06] A.M. Law. *Simulation Modeling and Analysis (Fourth Edition)*. McGraw-Hill, 2006.

[LGDL06] P. Le-Huy, S. Guérette, L.A. Dessaint, and H. Le-Huy. Dual-step real-time simulation of power electronic converters using an FPGA. In *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE'06)*, pages 1548–1553, Montréal, Canada, July 2006.

[Liu00] J.W.S. Liu. *Real-Time Systems*. Prentice Hall, Upper Saddle River, NJ, USA, 2000. ISBN 0-13-099651-3.

[LTH03] D. Lamberson, N.P. Teske, and J.K. Hedrick. A Model-Based Approach to the Implementation of Automotive Embedded Control Systems. In *Proceedings of the IEEE Intelligent Transportation Systems*, volume 1, pages 655–659, October 2003.

[MFL+05] A. Monti, H. Figueroa, S Lentijo, X. Wu, and R. Dougal. Interface issues in hardware-in-the-loop simulation. In *Proceedings of the Electric Ship Technologies Symposium*, pages 39–45. IEEE, July 2005.

[Mil98] D. Millinger. *Design and Implementation of a Communication Network Interface for a Fault-Tolerance Layer*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 3/3/182-1, 1040 Vienna, Austria, 1998.

[MIS04] The Motor Industry Software Reliability Association (MISRA). *MISRA-C:2004 Guidlines for the use of the C language in critical systems*, October 2004.

[NBAR04] S. Nabi, M. Balike, J. Allen, and K. Rzemien. An overview of hardware-in-the-loop testing systems at Visteon. SAE technical paper series, SAE International, 400 Commonwealth Drive, Warrendale, PA 15096-0001 USA, March 2004.

[NI03] National Instruments Corporation, 11500 North Mopac Expressway, Austin, TX 78759-3504, USA. *LabVIEW FPGA in Hardware-in-the-Loop Simulation Applications*, July 2003.

[NP99] J. Nikoukaran and R. Paul. Software selection for simulation in manufacturing: a review. *Simulation Practice and Theory*, 7(1):1–14, 1999.

[Nyq28] H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the A.I.E.E.*, 47:617–644, February 1928.

[OH06] R. Obermaisser and B. Huber. Model-based design of the communication system in an integrated architecture. In *Proceedings of the 18th International Conference on Parallel and Distributed Computing and Systems (PDCS'06)*, pages 96–107, Dallas, TX, USA, November 2006.

[OMG03] OMG. Smart Transducers Interface V1.0. Available Specification document number formal/2003-01-01, Object Management Group, Needham, MA, USA, January 2003. available at http://doc.omg.org/formal/2003-01-01.

[OP05] R. Obermaisser and P. Peti. Realization of virtual networks in the DECOS integrated architecture. In *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems 2006 (WPDRTS)*. IEEE, April 2005.

[OPHES06] R. Obermaisser, P. Peti, B. Huber, and C. El-Salloum. DECOS: An integrated time-triggered architecture. *e&i journal (Journal of the Austrian professional institution for electrical and information engineering)*, 3, March 2006.

[OS07] R. Obermaisser and M. Schlager. A simulation framework for virtual integration of integrated systems. In *Proceedings of the IEEE International Conference on "Computer as a tool" EUROCON*, Warsaw, Poland, September 2007.

[Pal00] R. Pallierer. *Validation of Distributed Algorithms in Time-Triggered Systems by Simulation.* PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 3/3/182-1, 1040 Vienna, Austria, 2000.

[Pet02] P. Peti. The concepts behind time, state, component, and interface – a literature survey. Technical Report 53/2002, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 1-3/182-1, 1040 Vienna, Austria, 2002.

[Pin95] J.J. Pinto. A neutral instrumentation vendor's perspective. In *ISA Proceedings '94 and Intech July '95*, July 1995.

[RP93] S. Radia and J. Pachl. Coherence in naming in distributed computing environments. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 83–92, Pittsburgh, PA, USA, May 1993.

[RS07] B. Rumpler and M. Schlager. Segmentation and conceptual chunking in embedded real-time system design. In *Proceedings of the ERCIM/DECOS Workshop on "Dependable Embedded Systems" at the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007) and the 10th EUROMICRO Conference on Digital Systems Design (DSD 2007)*, Lübeck, Germany, August 2007.

[RTC92] Radio Technical Commission for Aeronautics (RTCA) Inc., SC167, RTCA Secretariat, 1140 Connecticut Avenue, N.W., Suite 1020, Washington, DC 20036. *DO-178B/ED-12B Software Considerations in Airborne Systems and Equipment Certification*, December 1992.

[RTC00] Radio Technical Commission for Aeronautics (RTCA) Inc., SC180, RTCA Secretariat, 1140 Connecticut Avenue, N.W., Suite 1020, Washington, DC 20036. *DO-254 Design Assurance Guidance for Airborne Electronic Hardware*, April 2000.

[Rus01] J. Rushby. A comparison of bus architectures for safety-critical embedded systems. Technical report, Computer Science Laboratory, SRI International, September 2001.

[Rus02] J. Rushby. An overview of formal verification for the time-triggered architecture. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 2469 of *Lecture Notes in Computer Science*, pages 83–105, Oldenburg, Germany, September 2002. Springer-Verlag.

[Sch92] W. Schütz. *The Testability of Distributed Real-Time Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 3/3/182-1, 1040 Vienna, Austria, 1992.

[Sch95] W. Schütz. Testing distributed real-time systems: An overview. Technical Report 12/1995, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 1-3/182-1, 1040 Vienna, Austria, 1995.

[Sch03] M. Schlager. A simulation architecture for time-triggered transducer networks. In *Proceedings of the First Workshop on Intelligent Solutions for Embedded Systems (WISES'03)*, pages 39–49, Vienna, Austria, June 2003.

[SD02] D. Stang and R. Dandapani. An implementation of IEEE 1149.1 to avoid timing violations and other practical in-compliance improvements. In *Proceedings of the IEEE International Test Conference*, Baltimore, MD, USA, October 2002.

[SEW06] M. Schlager, W. Elmenreich, and I. Wenzel. Interface design for hardware-in-the-loop simulation. In *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE'06)*, pages 1554–1559, Montréal, Canada, July 2006.

[Sha75] R.E. Shannon. *Systems Simulation: The art and science*. Prentice Hall, Englewood Cliffs, New Jersey, 1975.

[SHW+06] M. Schlager, W. Herzner, A. Wolf, O. Gründonner, M. Rosenblattl, and E. Erkinger. Encapsulating application subsystems using the DECOS core OS. In *Proceedings of the 25th International Conference on Computer Safety, Security and Reliability (SAFECOMP)*, pages 386–397, Gdansk, Poland, September 2006.

[Sim96] H.A. Simon. *The Sciences of the Artificial*. MIT Press, 1996.

[Sma04] I. Smaili. Monitoring and debugging of real-time systems: A survey. Technical Report 17/2004, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 1-3/182-1, 1040 Vienna, Austria, 2004.

[SOE07] M. Schlager, R. Obermaisser, and W. Elmenreich. A framework for hardware-in-the-loop testing of an integrated architecture. In *Proceedings of the 5th IFIP Workshop of Software Technologies for Future Embedded & Ubiquitous Systems (SEUS)*, Santorini Island, Greece, May 2007.

[SS66] A.P. Sage and S.L. Smith. Real-time digital simulation for systems control. In *Proceedings of the IEEE*, pages 1802–1812, December 1966.

[SWE06] F. Skopik, M. Wihsböck, and W. Elmenreich. Anbindung einer Regelstreckensimulation an ein zu prüfendes System mittels eines Smart Inverted Transducers. Technical Report 2/2006, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 1-3/182-1, 1040 Vienna, Austria, 2006.

[SWH95] N. Suri, C.J. Walter, and M.M. Hugue. *Advances In Ultra-Dependable Distributed Systems*, chapter 1. IEEE Computer Society Press, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264, USA, 1995.

[TS01] A.S. Tanenbaum and M.V. Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[TTA02] TTA Group. Specification of the Time-Triggered Protocol TTP/C. Technical report, TTA Group, Schönbrunner Straße 7, A-1040 Vienna, Austria, July 2002. Specification edition 1.0.0, Available at http://www.ttpforum.org.

[TTC06] TTChip Entwicklungsgesellschaft mbH, Schönbrunner Strasse 7, A-1040 Vienna, Austria. *AS8202NF TTP-C2NF Communication Controller, Data Sheet, Rev.1.6*, July 2006.

[VDA03] Verband der Automobilindustrie (VDA). *HAWK2015 – Herausforderung Automobile Wertschöpfungskette*, 2003.

[Vee06] E. Veenendaal. Standard glossary of terms used in software testing. Technical report, Glossary Working Party, American Software Testing Qualifications Board, Inc. (ASTQB), June 2006. Available at http://www.astqb.org.

[Wen02] H.C. Wense. LIN specification package. Technical report, Motorola GmbH, Audi AG, BMW AG, DaimlerChrysler AG, Volcano Communications Technologies AB, Volkswagen AG, Volvo Car Corporation, Schatzbogen 7, 81829 Munich, Germany, December 2002. Revision 1.3.

[Wie02] T. Wiedemann. Next generation simulation environments founded on open source software and XML-based standard interfaces. In *Proceedings of the 2002 Winter Simulation Conference*, pages 623–628, 2002.

[Wik07a] Wikipedia. Determinism — Wikipedia, The Free Encyclopedia, July 2007.

[Wik07b] Wikipedia. Static testing — Wikipedia, The Free Encyclopedia, July 2007.

[Wik07c] Wikipedia. Verification and Validation — Wikipedia, The Free Encyclopedia, July 2007.

[Wir97] B. Wirz. Technical specifications for 600 series instrument grade electrostatic transducer. Available at http://controls.ae.gatech.edu, 1997.

[WLD+05] X. Wu, S. Lentijo, A. Deshmuk, A. Monti, and F. Ponci. Design and implementation of a power-hardware-in-the-loop interface: A nonlinear load case study. In *Applied Power Electronics Conference and Exposition (APEC'05)*, pages 1332–1338. IEEE, March 2005.

[Woj99] G. Wojciech. Hardware-in-the-loop simulation and its application in control education. In *29th ASEE/IEEE Frontiers in Education Conference*, pages 12b6–7–12b6–12, San Juan, Puerto Rico, November 1999. IEEE.

[ZLD+04] J. Zhenhua, R. Leonard, R. Dougal, H. Figueroa, and A. Monti. Processor-in-the-loop simulation, real-time hardware-in-the-loop testing, and hardware validation of a digitally-controlled, fuel-cell powered battery-charging station. In *IEEE 35th Annual Power Electronics Specialists Conference*, pages 2251–2257, June 2004.

[ZSL+04] C. Zavala, P. Sanketi, D. Lamberson, A.R. Girard, and J.K. Hedrick. Model-Based Real-Time Embedded Control Software for Automotive Torque Management. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, Le Royal Meridien, King Edward, Toronto, Canada, May 2004.

List of Publications

[1] M. Schlager. A simulation architecture for time-triggered transducer networks. In *Proceedings of the First Workshop on Intelligent Solutions for Embedded Systems (WISES'03)*, pages 39–49, Vienna, Austria, June 2003.

[2] M. Jakovljevic, M. Schlager, M. Plankensteiner, and S. Poledna. A path to mature safety-relevant automotive electronic solutions. *Automotive Electronics*, 1, March 2004.

[3] W. Elmenreich, S. Pitzek, and M. Schlager. Modeling distributed embedded applications on an interface file system. In *Proceedings of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 175–182, May 2004.

[4] W. Elmenreich and M. Schlager. Simulation-based development of embedded sensor fusion applications. In *Proceedings of the 2nd IEEE International Conference on Computational Cybernetics*, pages 147–153, September 2004.

[5] T. Losert, W. Elmenreich, and M. Schlager. Semi-automatic compensation of the propagation-delay in fault-tolerant systems. In *Proceedings of the Third IASTED International Conference on Communications, Internet, and Information Technology (CIIT 2004)*, pages 455–460, St. Thomas, VI, USA, November 2004. ISBN 0-88986-445-4.

[6] M. Schlager, M. Plankensteiner, and T. Albrecht. Zukunft bringt integrierte Architekturlösungen. *Elektronik Automotive*, pages 78–81, January 2005.

[7] T. Losert, M. Schlager, and W. Elmenreich. Fault-tolerant compensation of the propagation delay for hard real-time systems. *International Journal of Advanced Computational Intelligence & Intelligent Informatics (JACIII)*, July 2005.

[8] M. Schlager, E. Erkinger, W. Elmenreich, and T. Losert. Benefits and implications of the DECOS encapsulation approach. In *Proceedings of the 8th International IEEE Conference on Intelligent Transportation Systems*, pages 13–18, Vienna, Austria, September 2005.

[9] I. Wenzel, R. Kirner, M. Schlager, B. Rieder, and B. Huber. Impact of dependable software development guidlines on timing analysis. In *Proceedings of the IEEE International Conference on "Computer as a tool" EUROCON*, Belgrade, Serbia & Montenegro, November 2005.

[10] M. Schlager, W. Elmenreich, and I. Wenzel. Interface design for hardware-in-the-loop simulation. *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE06)*, pages 1554–1559, July 2006.

[11] M. Schlager, W. Herzner, A. Wolf, O. Gründonner, M. Rosenblattl, and E. Erkinger. Encapsulating application subsystems using the DECOS Core OS. In *Proceedings of the 25th International Conference on Computer Safety, Security and Reliability (SAFECOMP)*, pages 386–397, Gdansk, Poland, September 2006.

[12] W. Herzner, M. Schlager, T. Le Sergent, B. Huber, S. Islam, N. Suri, and A. Balogh. From model-based design to deployment of integrated, embedded, real-time systems: The DECOS tool-chain. In *Proceedings of the international DECOS Workshop at the Mikroelektroniktagung 2006*, Vienna, Austria, October 2006.

[13] M. Schlager, R. Obermaisser, and W. Elmenreich. A framework for hardware-in-the-loop testing of an integrated architecture. In *Proceedings of the 5th IFIP Workshop of Software Technologies for Future Embedded & Ubiquitous Systems (SEUS)*, Santorini Island, Greece, May 2007.

[14] H. Eriksson, J. Vinter, B. Leiner, and M. Schlager. Towards a DECOS fault injection platform for time-triggered systems. In *Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN)*, Vienna, Austria, July 2007.

[15] B. Rumpler and M. Schlager. Segmentation and conceptual chunking in embedded real-time system design. In *Proceedings of the ERCIM/DECOS Workshop on "Dependable Embedded Systems" at the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007) and the 10th EUROMICRO Conference on Digital Systems Design (DSD 2007)*, Lübeck, Germany, August 2007.

[16] R. Obermaisser and M. Schlager. A simulation framework for virtual integration of integrated systems. In *Proceedings of the IEEE International Conference on "Computer as a tool" EUROCON*, Warsaw, Poland, September 2007.

[17] W. Herzner, R. Schlick, M. Schlager, B. Leiner, B. Huber, A. Balogh, G. Csertan, A. LeGuennec, Th. LeSergent, N. Suri, and S. Islam. Model-based development of distributed embedded real-time systems with the DECOS tool-chain. In *Proceedings of the SAE AeroTech Congress & Exhibition*, Los Angeles, CA, USA, September 2007.

[18] J. Vinter, H. Eriksson, A. Ademaj, B. Leiner, and M. Schlager. Experimental evaluation of the DECOS fault-tolerant communication layer. In *Proceedings of the 26th International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, Nuremberg, Germany, September 2007.

[19] B. Leiner, M. Schlager, R. Obermaisser, and B. Huber. A comparison of partitioning operating systems for integrated systems. In *Proceedings of the 26th International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, Nuremberg, Germany, September 2007.

Curriculum Vitae

Martin Schlager

March 4^{th}, 1978	Born in Vienna, Austria
September 1984 – June 1988	Elementary School in Schottwien
September 1988 – May 1996	Comprehensive Secondary School ("AHS") in Neunkirchen
October 1996 – May 1997	Military Service in Mautern and Wr. Neustadt
October 1997 – May 2002	Studies of Computer Science at the Vienna University of Technology
June 6^{th}, 2002	Graduation with distinction Master's Degree ("Dipl.-Ing.")
September 2004 – March 2005	Studies of Computer Science Management at the Vienna University of Technology
April 5^{th}, 2005	Graduation with distinction Master's Degree ("Mag.rer.soc.oec.")
June 2002 – ongoing	R&D Consulting and Project Coordination at TTTech Computertechnik AG, Vienna
September 2002 – ongoing	PhD Studies at the Institute of Computer Engineering, Vienna University of Technology, supported by the Austrian Academy of Sciences (DOC program)