Dipl.-Ing. Thomas Wolfgang Pieber, BSc

Stimuli Creation from Use Cases: An Approach to Connect Environment and Hardware Simulations

DOCTORAL THESIS

to achieve the university degree of

Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Inform. Dr.sc.ETH Kay Uwe Römer

Advisor
Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger

Institute for Technical Informatics

Graz, July 2020

Abstract

Simulations help us to develop new hardware for the future. As the development of new hardware needs to get faster and result in fewer errors, new approaches to perform the simulations need to be developed. This thesis explores the idea to generate the stimuli for the hardware simulations based on the simulation of the intended use cases.

During the work at this thesis two core questions are asked and answered. The first question is how the stimuli can be created from a description of the use case and how the created stimuli can be fed to the hardware simulation. The second question deals with the speed of the simulation. As hardware simulations can be slow, and additionally simulating the intended use cases further reduces the simulation speed, we seek new methods to accelerate the simulation.

The developed concept is based on the co-simulation of the hardware and the intended environment. The use cases for the new hardware specify the parameters for the environment simulation. Inside this virtual environment one or multiple instances of the new hardware are placed and connected to the hardware simulation. The environment simulation is then able to generate the effects of the environment on the new hardware. These effects can be produced by any property that is generated by the environment itself such as gravity or light, or can be produced by data that is transmitted through the environment to communicate with the new hardware. In addition to that, the new hardware is also able to communicate to other entities in the environment by using attached actuators or communication interfaces.
To test the developed concept, the Gazebo simulator, a simulation tool for robotic purposes, is used to simulate the environment. The hardware simulation is performed using the SystemC language. This allows the simulation of the hardware at any layer of abstraction.
As an example, a co-simulation is considered, where a sensor platform can be accessed by a robot using a wireless communication interface.
The hardware simulation can experience large amounts of idle time when also simulating the environment of the hardware. This can happen when, for example, an entity moves in the environment. Such events can go unnoticed by the hardware. For such cases, acceleration methods for the hardware simulations have been created. These methods work by checking the received input data. The information gained by observing the input data is combined with the knowledge about the internal state of the hardware to decide whether the full simulation needs to be executed or if the resulting changes can be estimated. To minimize errors due to non-linearities of the full model that are not represented in the estimation, the estimation process can also trigger the full simulation.
These methods can not only accelerate the overall simulation process, but can also reduce

the memory requirements needed to store the created trace-files.

Two sensor prototypes have been created to generate data for the simulations and help verifying the simulation concept. These prototypes are able to generate data on the energy output of energy harvesting methods and the energy usage of the components of a smart sensor. Furthermore, the testbench for one of the prototypes is able to emulate the behaviour of various energy sources.

As this thesis was funded by the IoSense project, the developed simulation concepts and the hardware prototypes have been used by project partners in order to examine new sensor hardware.

Kurzfassung

Simulationen helfen uns bei der Entwicklung neuer Hardware für die Zukunft. Da die Entwicklung von neuer Hardware schneller und trotzdem weniger fehlerhaft sein soll, müssen neue Ansätze zur Durchführung der Simulationen entwickelt werden. Diese Dissertation ging der Idee nach, die Stimuli für die Hardwaresimulationen auf Basis der vorgesehenen Anwendungsfälle zu generieren. Während der Arbeit an dieser Dissertation wurden zwei Kernfragen gestellt und beantwortet. Die erste Frage ist, wie die Stimuli durch eine Beschreibung der Anwendungsfälle generiert werden können und wie die erzeugten Stimuli der Hardwaresimulation zugeführt werden können. Die zweite Frage beschäftigt sich mit der Geschwindigkeit der Simulation. Da Hardwaresimulationen langsam sein können und die zusätzliche Simulation der Anwendungsfälle die Simulationsgeschwindigkeit weiter verringert, wird im Zuge dieser Frage geklärt, welche Methoden zur Beschleunigung der Simulation es gibt.

Das entwickelte Konzept basiert auf der Kosimulation der Hardware und der geplanten Umgebung. Die Anwendungsfälle für die neue Hardware spezifizieren die Parameter für die Simulation der Umgebung. In dieser virtuellen Umgebung werden ein oder mehrere Exemplare der neuen Hardware platziert und mit der Hardwaresimulation verbunden. Die Umgebungssimulation kann dann die Effekte der Umgebung auf die Hardware generieren. Diese Effekte können durch jede Eigenschaft der Umgebung selbst, also Gravitation oder Lichteinfall, oder durch Daten, die durch die Umgebung gesendet werden um mit der neuen Hardware zu kommunizieren, verursacht werden. Zusätzlich ist die neue Hardware in der Lage mit anderen Entitäten in der Umgebung zu kommunizieren. Dies kann durch die Verwendung von angebrachten Aktuatoren oder Kommunikationsschnittstellen geschehen. Um das entwickelte Konzept zu testen wurde der Gazebo Simulator, ein Simulationswerkzeug für die Robotik, verwendet um die Umgebung der neuen Hardware zu simulieren. Die Hardwaresimulation wurde in der SystemC-Sprache geschrieben. Dies erlaubt die Simulation der Hardware auf unterschiedlichen Abstraktionsebenen. Beispielhaft wurde die Kosimulation einer Sensorplattform untersucht welche von einem Roboter über eine kabellose Kommunikationsschnittstelle kontaktiert werden kann.
Die Hardwaresimulation kann langen Wartezeiten ausgesetzt sein wenn die Umbebung der Hardware mitsimuliert wird. Dies kann beispielsweise der Fall sein wenn sich eine Entität in der Umgebung bewegt. Ein solches Event kann von der Hardware übersehen werden. Für solche Fälle wurden Methoden zur Beschleunigung der Hardwaresimulation erstellt. Diese Methoden zur Beschleunigung überprüfen die empfangenen Eingaben der Simulation um die Simulation zu beschleunigen. Die gewonnene Information aus dem Beobachten der Eingaben wird mit dem Wissen um den internen Zustand der Simulation verbunden um zu ermitteln ob die komplette Simulation ausgeführt werden muss oder ob die resultierenden Änderungen abgeschätzt werden können. Um die Fehler der Abschätzung, die durch

in der Abschätzung nicht berücksichtigte Nichtlinearitäten entstehen, zu minimieren kann der Abschätzungsprozess auch die vollständige Simulation anstarten.
Diese Methoden können nicht nur den Gesamtprozess der Simulation beschleunigen sondern auch die Anforderungen an den Speicher reduzieren, um die erstellten Trace-Dateien zu speichern.

Es wurden zwei Sensor Prototypen entwickelt um die Daten für die Simulation zu generieren und um das Simulationskonzept zu verifizieren. Diese Prototypen sind in der Lage Daten über die Energieausbeute von Energiegewinnungsmethoden und den Energieverbrauch der Komponenten eines intelligenten Sensors zu generieren. Desweiteren ist ein Prüfstand der Prototypen dazu in der Lage, das Verhalten von verschiedenen Energiequellen zu emulieren.

Da diese Dissertation durch das IoSense Projekt finanziert wurde, wurden die entwickelten Simulationskonzepte und die Prototypen von Projektpartnern dazu verwendet um neue Sensorkomponenten zu testen.

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date	Signature

Note of Thanks

Herewith I would like to thank all those who have guided me in the course of this doctoral thesis.

I want to express my thankfulness to Associate Prof. Dipl.-Ing. Dr. Christian Steger, who has tutored me while conducting this work. His professional and personal support guided me throughout this time. The comments of my supervisor, Univ.-Prof. Dipl.-Inform. Dr.sc.ETH Kay Uwe Römer, have significantly improved the quality of this thesis. For this task he deserves the biggest thanks. Dipl.-Ing. Holger Bock's skilful assistance and Andreas Wallner's competent help have significantly contributed to improving the experience of this work.

A special thank you goes to the students Fikret Basic and Benjamin Mößlang, who provided some results for my thesis and helped by conducting their theses with me.

Beyond that I want to thank my parents Josef and Edith and my brothers Daniel and Roman, who enabled me to participate in this study and encouraged me throughout it. Their support has led me through these tough times.

Furthermore, I want to thank Natalie, my girlfriend, for her patience and the support she gave me. Without her the experience of conducting this thesis would not have been the same.

Another special thank you goes to my fellow students Jakob, Marco, Sarah, and Thomas, who also supported me on countless occasions throughout my university time. The discussions with them have inspired and brightened my time, I found true friends in them. Especially I want to thank Thomas, with whom I also shared the office during the course of this thesis and conducted his researches in the same project.

Last but not least I want to thank all my friends who patiently supported me.

Contents

List of Figures

List Of Abbreviations

3D	Three Dimensional
ACSL	Advanced Continuous Simulation Language
CPS	Cyber Physical System
CRV	Constrained Random Verification
CS	Continuous Simulation
DES	Discrete Event Simulation
FMI	Functional Mockup Interface
GNS3	Graphical Network Simulator-3
GPIO	General Purpose Input Output
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HiL	Hardware in the Loop
IDSSO	Input Dependent Simulation Speed Optimization
IoT	Internet of Things
JSON	Java Script Object Notation
LBNL	Lawrence Berkeley National Laboratory
MiL	Model in the Loop
NFC	Near Field Communication
NS-2	Network Simulator Version 2
NS-3	Network Simulator Version 3
OMNET++	Objective Modular Network Testbed in C++
PCB	Printed Circuit Board
PiL	Processor-in-the-Loop
RTF	Real-Time-Factor
RTL	Register Transfer Level
SC	Security Controller
SiL	Software-in-the-Loop
SIMCOS	Simulation of Continuous Systems
SSiL	System Simulation in the Loop
TLM	Transaction-Level Modelling
uC or μC	Micro Controller
VHDL	Very High Speed Integrated Circuit Hardware Description Language
ViL	Vehicle-in-the-Loop
WSN	Wireless Sensor Network
XiL	X in the Loop

XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

1

Introduction

The design of new hardware can require a lot of testing, especially if there is no re-use of already tested building blocks. To test the new designs, simulations are often used. There are many simulation tools available to test the functionality of a single piece of hardware. These simulation instances are written in Hardware Description Languages (HDLs). These languages, such as the Very High Speed Integrated Circuit HDL (VHDL)[1] or Verilog [2] describe the hardware in detail. The resulting specification, which describes the hardware, can be used to synthesize the final hardware layout which can then be manufactured. Other languages, such as SystemC [3], are designed to describe the hardware on a higher level of abstraction. This reduces the amount of work needed to create a decent simulation. Furthermore, as the description is more abstract, fewer calculations are required to simulate the behaviour. Thus the simulation is faster. This comes at the cost of increased development time if the created design is then to be synthesized.

All these simulation tools are designed to be used to simulate one piece of hardware. In the case of VHDL or Verilog one chip is simulated. Due to the higher abstraction, SystemC is being used to simulate a collection of components. To perform simulations that connect multiple such collections, other simulation methods need to be used.

There are some tools that can be used to simulate a collection of individual computing nodes. One of the most prevalent tools to simulate a network of computers is Simics [4]. Furthermore, simulation tools for Wireless Sensor Networks (WSNs)[5] can be used to estimate the connectivity between the nodes.

The primary use of these tools is to test different distributed algorithms. All those simulation tools sacrifice accuracy for speed [6]. This decrease in accuracy is acceptable for these applications as slightly different computation times or different energy consumption does not change the validity of the algorithm. These tools are, however, not intended to be used to validate the behaviour of the hardware itself.

There are further simulators used to simulate WSNs, such as OMNET++ [7], the LBNL network simulator (or NS-2 [8]), NS-3 [9], or GNS3 [10]. These simulators are designed to simulate the communication between the sensor nodes. To do so, these simulators use abstract models of the hardware. These simulators do either not care about the environment at all or use a simplified static environment as a reference.

1.1 Motivation

WSN nodes are the most basic instance of a Cyber Physical System (CPS). The United States National Science Foundation defines CPS as follows [11]:

> CPS are engineered systems that are built from, and depend upon, the seamless integration of computation and physical components. CPS tightly integrate computing devices, actuation and control, networking infrastructure, and sensing of the physical world.

Creating new hardware that is intended for the use in a CPS thus requires simulations of the hardware in their intended physical environment. As Ying and Sztipanovits stated in their report [6], new simulation tools are needed to fully model CPS hardware in their environment.

This problem also occurred in the IoSense [12] project. Here we needed a simulation that can estimate the effects on the lifetime of sensors caused by hardware and software changes. These sensors are powered, read, and configured by using NFC technology. This puts forward the problem that the estimations need to be very accurate as charging an overly large capacitor is inefficient.

This combination of problems, the lack of simulation tools for CPS and the project that needs a very accurate simulation of a CPS in the corresponding environment led to the research that culminated in this thesis.

1.2 Contribution

In this thesis a new concept to model and simulate CPSs is being explored. An overview about this thesis is given in Figure 1.1.
As the simulation of the CPSs should be excited by stimuli created from use cases, a proper definition of the terms “stimulate” and “use case” is needed. In the “IEEE standard glossary of modeling and simulation terminology” the term “stimulate” is defined in the following way [13]

> To provide input to a system in order to observe or evaluate the system’s response.

The term “use case” is defined by the IEEE 26515:2018 standard as the following [14]

> Description of behavioural requirements of a system and its interaction with a user.

The work of this thesis starts at the top of this image: *State of the art analysis of hardware systems simulations.* Here the focus is on the *security* and*efficiency* of new hardware. This work resulted in a first publication (*P1*). The next step is the analysis of the *creation of the stimuli for hardware simulations.* At this point the research questions of this thesis have been defined as: *How can the stimuli for the hardware simulation be created using the use case description,* and *how can the simulation be made more efficient?* The

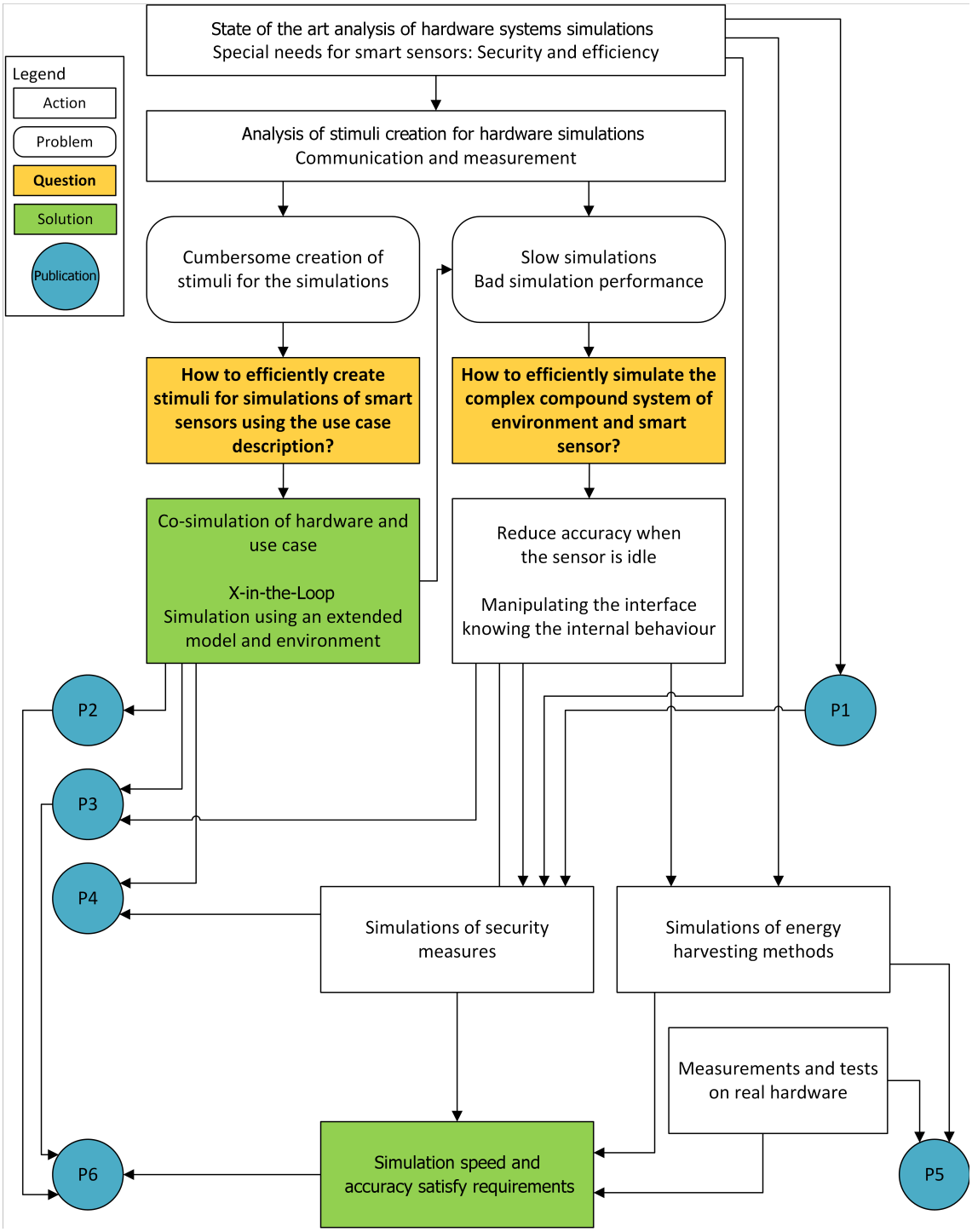

Figure 1.1: Overview of this thesis. It starts by analyzing the state of the art regarding simulations of hardware. Then two research questions (yellow boxes) are formed from occurring problems. The solutions to these questions are finalized in the green boxes. During this thesis six publications (P1 - P6 in blue circles) have been created.

answers to these questions are shown in the green boxes. During the process of answering the research questions more publications (*P2* - *P6*) have been created.

To solve the problem of creating a tool that can efficiently simulate the behaviour of a CPS while also simulating the hardware itself, it is of utmost importance to understand the basic concepts of hardware simulation. Stimuli of the surroundings need to be fed into the simulation of the hardware. This is usually done by using precomputed stimuli. In the case of simulating a sensor and actuator network this is not possible, as the input of a sensor circularly depends on the output of an actuator. Thus, the stimuli need to be generated inside the simulation environment.

The calculation of stimuli for a hardware simulation poses two problems:

1. How can the stimuli be calculated automatically if a description of the use-case is available?

2. How can the simulation speed be improved while maintaining the accuracy of the simulation where it is needed?

To generate stimuli for a hardware simulation that depend on the output of the simulation itself, a co-simulation of the hardware and the use case can be performed. In this co-simulation an initial stimulus can be created that is projected onto the hardware simulation. When the calculations of the hardware reaction are completed, the outputs can be used to compute the reaction of the environment that represents the use case. This reaction leads to an update of the stimuli on the hardware. This approach can also be used if multiple hardware instances are situated in the simulated environment. In the case of a sensor simulation, a three dimensional (3D) environment can be used. This allows the placement of virtual sensors in the environment, taking measurements of the surroundings, and represent effects of the sensors actions that in turn can be measured again. In other words, a simulation similar to a "Hardware in the Loop (HiL) simulation" is created where also the hardware itself is simulated. This differs from a so called "Model in the Loop (MiL) simulation" [15] as in the MiL simulation the model only functionally represents the intended hardware. Brockmeyer et al. state in their publication that the used hardware model in a MiL simulation is usually a crude reference model of the intended final hardware [16]. Thus, the SystemC simulation used in this thesis represents an extended and refined version of such a hardware model that can already be used to execute code written for this hardware. Such simulation can be called a "System Simulation in the Loop (SSiL)".
The overhead in computation imposed by this simulation technique further reduces the simulation speed as more calculations need to be performed to achieve the same result. Question two is therefore equally important to be answered.

As other simulators that compute the behaviour of multiple computers show, the simulation speed can be improved by reducing the accuracy of the simulation [17]. One possibility where the accuracy can be reduced is during the phases when a sensor is idle. During these phases only the volatile values, such as the idle energy consumption, need to be estimated. Further simulation speed improvements can be achieved by combining multiple estimation steps or pausing the simulation if the sensor becomes idle during the simulation step. So, this thesis explores the possibility to monitor the stimuli of the simulation to decide if the

simulation of the next step will cause significant changes to the overall system.
Such improvements can benefit the simulation speed while the reduction in accuracy is minimal.

To test the concepts developed in this thesis, an example simulation has been written. Using this simulation, we are able to derive the errors that are introduced by the co-simulation, as well as the speed improvements.

1.3 Outline

In Chapter 2 background information about the simulation of sensors is presented. Furthermore, the work of other researchers are examined, and significant differences to our work are highlighted. Chapter 3 focuses on the design of the simulator. This chapter starts by defining requirements for the simulator. Furthermore, the generation of stimuli in the designed co-simulation is evaluated. A major topic in this chapter is the optimization of the simulation to achieve useful simulation speeds. Crucial steps of the implementation are explained in Chapter 4. This chapter focuses on the realization of the design. To do so, requirements are formulated and implemented. Furthermore, this chapter defines an example simulation as a proof of concept that is used to test the simulator. The results of the proof-of-concept simulation and the evaluation of the results are shown in Chapter 5. Possible improvements to the current state of the simulator are explained in Chapter 6. A conclusion of this thesis is also given in this chapter. Acknowledgements to the sponsors and partners without which this thesis would not have been possible are given in Chapter 7.

In appendix A the publications that have been created in the course of this thesis are attached.

2
Background and Related Work

This chapter focuses on the background upon which this thesis builds. General information about simulations, sensor efficiency, and sensor security are presented. Furthermore, the work of other researchers that have studied and are studying this field are presented and compared to this thesis.

2.1 Background

This section provides information about techniques used throughout this thesis. As this thesis revolves around the topic of simulations, general information about this topic is presented here. Furthermore, a part of this section is dedicated to the simulation tools which are used to implement the developed concepts.

It is a fact that there are many types of simulations, which are used to predict the behaviour of an arbitrary system or process without disrupting the actual system or the need to create the system before the experiments [18]. To be able to do this, a model of the desired system needs to be created. In the case of a user who wants to create a realistic model of an already existing system, the simulation results are compared to the real system and then the model is refined. If the user wants to create a new system that should fulfil defined requirements, the simulations show if the requirements are met. Should the requirements not be met, the analysis of the simulation allows the refinement of the new system in order to get closer to the goal. There are simulations for any kind of system such as chemical systems [19], electrical systems [1–3, 20], and robotic systems [21–24]. In the course of this thesis electrical and robotic systems are examined in greater detail.

Simulations have two core measures that can be improved or traded for each other: accuracy and speed [25, 26]. If simulations need to be more accurate, more calculations are needed to be done and therefore the simulation speed is slower. To find a good balance between these two core measures is the core issue when optimizing a simulation.

This thesis explores the possibility to devise a simulation where not only one part (hardware or environment) is simulated, but both with distinct simulators. To realize such an approach this thesis uses SystemC [3] to simulate the hardware and the Gazebo simulator [21] to provide an environment simulation.

SystemC is chosen as it is able to simulate a system on a high layer of abstraction and is therefore able to achieve a high simulation speed. The Gazebo simulator is widely known in the community of robotics. As its functionality can be expanded by implementing

plugins, a wide array of physical properties can be supported.

Simulation Types

There are two main groups of simulations: "*Continuous simulations*" (CSs) and "*Discrete event simulations*" (DESs).

Continuous simulations can be written with languages such as the "*Advanced Continuous Simulation Language*" (ACSL) [27] or "*Simulation of Continuous Systems*" SIMCOS [28, 29].
These simulations typically use differential equations to describe a system that is then simulated. By solving the set of given differential equations the solution of the simulation is calculated.

For systems that cannot be easily modelled using differential equations, discrete event simulations (or event-based simulations) are used. DESs change their simulation state at discrete points in time. Mathematically this corresponds to a step-function [30]. This assumes that the simulation state does not change between the simulation steps. When using sufficiently small simulation steps, accurate results can be generated.

SystemC

SystemC [3] is an expansion of C++ to allow the description of hardware. To do so, it adds a notion of time to C++ and introduces a method to compute parallel tasks [31]. It is an event-based simulation that supports the simulation of hardware on different layers of abstraction.
The communication between the modules of the simulation is based on logical signals and channels. Each module needs to provide ports to transmit data in and out of the module. A module can be nested in another module. Nested modules are connected using a channel. A complete description and tutorial on the use of SystemC has been written by Black et al. [32].
In a typical SystemC simulation the testbench provides the master clock for the simulated components and represents the environment where the stimuli are created. The testbench furthermore monitors channels in the modules of the simulation to create a trace-file for the user.
SystemC is used in this thesis because of its ability to simulate the hardware on a high layer of abstraction.

Gazebo

Gazebo is a simulation tool developed by the Open Source Robotics Foundation [21]. It is a simulation tool for robotic purposes. It operates in discrete time steps. Each module of the simulation needs to implement a function that is called upon every time step. As the modules are called after each other and the core of Gazebo waits for each module to finish its calculations, the complete simulation operates sequentially.
The communication between the modules of the simulation is based on a publish-subscribe pattern. The simulation core provides the data structures necessary to provide this communication. Each module can publish data to the topics and can subscribe to the topics

it wants information about.
These simulations typically contain one or more robots that move through the environment. This provides the user with the possibility to check a robot's behaviour in certain conditions.
Gazebo was used in this thesis because of its extensibility. By implementing plugins various new features can be added. Furthermore, due to the large user community, there are a lot of plugins available that help in the development process.

Stimuli of Simulations

Stimuli are required to inflict change on a system and observe its reaction to it. In classical hardware simulation such stimuli are either generated manually to test the hardware reaction in a certain use case, or the stimuli are generated randomly using a Constrained-Random-Verification (CRV) technique. CRV is used to randomly test stimuli examples in the stimuli-space. There are some methods that try to improve the speed of CRV methods [33–36].
As CPS are interacting with their environment and expect plausible inputs, CRV is not useful to test these systems. Therefore, also the environment needs to be simulated in conjunction with the CPS. Fummi et al. created a simulation of an embedded system that interacts with the environment [37]. In their paper the authors present a co-simulation of SystemC and NS-2. In their case the focus lies on the simulation of the available bandwith for applications with respect to different protocols.
In the case of MiL simulations, the stimuli can be generated by the intended surrounding itself, or a functional model of the surrounding [16]. In this work, the stimuli are generated by a model of the environment.

2.2 Related Work

This section compares the work of this thesis to already existing work found in the literature. As this work can be analyzed from different perspectives, this section also illustrates the different aspects of this work.

2.2.1 Co-Simulations

Co-simulation is defined as the coordinated execution of two or more models that differ in their representation as well as in their runtime environment [38]. Steinbrink et al. state that the definition of Schloegl et al. implies some sort of interaction between hardware and software. But if the purpose of the co-simulation is to conduct the testing of the hardware it is a HiL simulation [39]. The authors furthermore state that the benefit of a co-simulation is the separation of the modelling and simulation tasks. Thus different teams can cooperate in creating the co-simulation. Each team provides input to the simulation based on their core expertise. This then enables the modelling and simulation of large complex systems and so-called "systems of systems".

The creation and orchestration of such large simulations is a major task when creating co-simulations. To do so the Functional Mockup Interface (FMI) [40] can be used [41]. During the conduction of this thesis FMI interfaces have been developed for SystemC [42] and Gazebo [43].

In the FMI definition for co-simulations every solver (or simulator) is a slave to the FMI-master. That implies that the master defines a time interval for which the simulators need to do the simulation. At the end of this step the generated information is transmitted to the intended recipient simulator. To use the FMI for a co-simulation, the simulation needs to be built with that in mind.
In contrast to that, the implementation of this thesis incorporates the "master" side of the communication in the environment in which the sub-simulation takes place. In the example used in this thesis the master simulation is the Gazebo simulation calculating the environment, and the slave simulation is the SystemC simulation that receives stimuli from the environment and returns its result to the environment. The approach in this thesis only requires that the testbench supports the in- and output of information to any other simulation.

Another framework that is used to create co-simulations of CPS has been proposed by Zhang et al. [44]. This framework also requires that all simulation components are created with the final goal of connecting them. Also in this framework the synchronization of the component simulations is done either by the communication controller or the clocks run independent of each other.

2.2.2 X-in-the-Loop Simulations

There is a lot of research in the literature about X-in-the-Loop (XiL) simulations. Most of the research focuses on the creation of better HiL simulations [45–47]. According to Brockmeyer et al. the HiL simulations are to be done after the other XiL simulations [16]. This is due to the fact that a functioning hardware needs to exist before the HiL simulations can start.

MiL simulation is not that prominent in the literature [15, 48]. Most MiL simulations have been created for the automotive industry.

XiL simulations, other than MiL and HiL, that occur in the literature are Software-in-the-Loop (SiL) and Processor-in-the-Loop (PiL). These XiL simulations can be seen as a transition from MiL to HiL. In an SiL simulation the Software that is intended to be executed in the new hardware is tested. The PiL simulation is testing the single components of the final hardware.
As an expansion to the HiL simulation some researchers created simulations that embed the complete hardware system in a simulation of the environment. One example of such simulations are Vehicle-in-the-Loop (ViL) simulations [49, 50].

Model-in-the-Loop Simulations

In this thesis an MiL simulation is used as a starting point. From there the concept is expanded to use more refined models of hardware and a complete simulation of the environment.

Classic MiL simulations use a functional model of the hardware under development and a simplified model of the environment to stimulate the hardware simulation [15]. Thus, it can be said that the classic MiL simulation is performed in the early development phases. In later phases the model of the MiL simulation is replaced by more advanced components such as the complete hardware in the end (HiL). Currently there are no MiL simulations in the literature where a detailed model of the hardware is created and used in these simulations.

Synchronization of XiL Simulations

The topic of synchronizing co-simulations is common in the literature [51–56]. One of the most used methods to synchronize any parallel system is using a lock-step pattern (Figure 2.1).

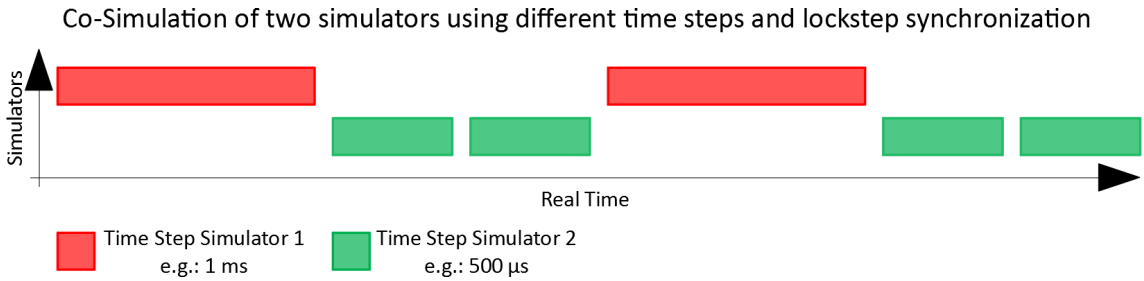

Figure 2.1: Timeline of a lock-step co-simulation. The execution of both simulators appears single-threaded. Simulator 1 advances one time step; Simulator 2 advances until the simulation times match.

This synchronizes the simulations after every time step. This ensures that all simulations have the same simulation time at the end of every step. Thus, all messages that are exchanged between the simulations are sent and received at the same simulation time.

A proper synchronization is of utmost importance in XiL simulations. Therefore, also in this thesis, a lock-step approach is used. As the lock-step approach effectively eliminates parallelism, slight changes in the execution need to be made to speed up the co-simulation.

2.2.3 Stimuli Generation for Simulations

For many simulations of hardware designs, stimuli are created at random but with certain constraints. This constraint-based random verification performs well to find border cases for the hardware and test hardware features [57, 58]. Thus, most of the literature that is concerned with the generation of stimuli for simulations tries to find better constraints for this kind of testing [33, 59].

The testing and verification process for more complex systems not only requires the hardware to work properly but also that the software run on that hardware is correct [60, 61]. Thus, normal constraint-based testing is not enough.

To fully test the complete design of hardware and software, a simulation of the complete system needs to be performed.

2.2.4 SystemC Co-simulation

There are a number of researchers that have used SystemC as a component for co-simulations [62–66]. These publications connect SystemC to either other HDLs or to a system-simulation tool such as Matlab or Simulink. These examples of co-simulation in combination with SystemC show different methods to connect another simulator to SystemC.

Simulators	Connection	Synchronization
SystemC - NS-2; [37]	Network	After every time step
SystemC - Simulink; [62]	Not specified	When the input is modified or at predictable events
SystemC - Simulink; [63]	Network	After every time step
SystemC - VHDL; [64]	Wrapper methods	Continuously
SystemC - Matlab; [65]	Network	At synchronization events
SystemC - SystemC; [66]	Not specified	At synchronization events
SystemC - Gazebo; This work	Pipes or Network	After every time step

Table 2.1: Comparison on SystemC co-simulations.

Fummi et al. [37] use simulation kernel modifications to keep the simulations synchronized. To do so, every message sent contains the simulation time when it was sent. In this way the communication between the simulations keeps the simulations synchronized.
Mendoza et al. [63] use a synchronization function whenever a transition between the Simulink and SystemC parts happens. As the authors of this paper include SystemC blocks in their Simulink model, the synchronization is controlled by Simulink. In their case multiple SystemC simulations can run in parallel with each other but do not necessarily share the same time.

In contrast to these examples of time synchronization, the approach in this thesis uses a lock-step execution to keep all simulators synchronized. To do so, a message is generated at the end of every time step of every simulator to keep the others informed of the status. In the case of the example simulation of this thesis, the Gazebo simulator sends a message to SystemC when the simulation step is finished, telling the SystemC simulator for how long to simulate. After that the Gazebo simulator is idle until it receives a message from SystemC that the simulation time is up-to-date. This does not require any modifications to the simulation kernels. Furthermore, if multiple SystemC simulations are executed in parallel, although they do not directly synchronize their clocks, all simulation clocks will be synchronized at the end of the time step.

2.2.5 Parallel SystemC

As the simulation speed of SystemC can be slow, a number of researchers tried to optimize the simulation tool to utilize the multiple cores of a modern computer. This is typically done by altering the standard scheduler of SystemC to allow multiple CPU cores or Graphics Processing Units (GPUs) to execute the simulation components [66–71].
There are three main approaches to have concurrent simulations:

1. **Multi-instance-one-simulation:** One simulation contains multiple parallel executed tasks.

2. **One-instance-multi-simulation:** One simulation computes one task. Multiple such simulations communicate with each other.

3. **Multi-instance-multi-simulation:** Multiple simulation instances communicate with each other. Each of these simulations contain multiple parallel executed tasks.

Work	Parallelism type
Concurrent SystemC simulations; [67]	1
Fast and Accurate SystemC; [66]	2
Parallel SystemC on SMP Workstations; [68]	2
parSC; [69]	2
Mixed-abstraction SystemC; [70]	2
Time-decoupled SystemC; [71]	2
Multiple SystemC instances; This work	2

Table 2.2: Comparison on the parallelization of SystemC.

As the basic SystemC is not parallel, the literature tries to achieve parallelism by updating the SystemC core to allow multiple parallel tasks (type 1) or combining multiple SystemC simulations to split a larger simulation (type 2). The combination of both would yield a parallel simulation of type 3.

In contrast to the other simulations, parallelism in this thesis means that multiple complete SystemC simulations are executed in parallel. Each of these simulations could also be executed sequentially to yield the same result, whereas the parallelism in the other approaches means that one single simulation is split into multiple smaller simulations that need to work together to produce the result. In other words, the approach of this thesis parallelizes multiple instances of SystemC simulations, whereas the other examples parallelize the simulation of the components of one instance of a SystemC simulation.

2.2.6 Network Simulations

When simulating an IoT (Internet of Things) network, one of the main challenges is the simulation of the network components. As many devices need to communicate with each other, such a simulation must be able to calculate the behaviour of all network nodes. There are a number of publications that describe such simulations. One of the widest known tools is Simics [4]. This simulation tool virtually connects the network nodes with each other. All the nodes are abstractly simulated and react to the incoming messages. The combination with another simulation tool such as SystemC is possible, although the simulation becomes very slow. Therefore Khan and Wolf needed to scale the SystemC clock and temporally decouple the Simics and SystemC simulations [72]. In addition to that, Khan et al. tried to optimize the Simics - SystemC co-simulation by creating a multi-threaded co-simulation [73].

Work	Stimuli generation	Synchronization
Simics; [4]	Logical interaction of modules	Internally synchronized; Externally temporally decoupled
SEED; [74]	Classic testbench	Event based
Internet of Simulation; [75]	N/A	N/A
Simulation of node and environment; This work	From 3D environment	At the end of every step

Table 2.3: Comparison of related work performing simulations of networks.

Simics and SEED are designed to simulate a network of multiple communication partners. Therefore, their priority is to accurately describe the logical connection between the components. The main aspect of this thesis is the integration of a hardware simulation in an environment. If multiple instances of hardware simulations are placed inside the same environment, the simulation of a network comes as an emergent feature.

2.2.7 Speed vs. Accuracy of Simulations

The speed of a simulation is a very important criterion when comparing different simulations. More accurate simulations are usually slower than simulations that are written on higher levels of abstraction. MathWorks describes methods to balance the simulation speed and simulation accuracy [25]. Here three main methods for balancing the speed and accuracy are given:

1. **Change the model fidelity or scope:** When the model fidelity or the simulation scope is decreased, fewer calculations need to be performed.

2. **Change the sample time:** When increasing the sample time (or time step) longer durations can be simulated. This decreases the temporal accuracy of the simulation.

3. **Change the number of solver iterations:** For simulations that solve numerical problems by iteratively improving the result, a decrease in these iterations reduces the accuracy but speeds up the simulation. This method is not applicable to SystemC simulations.

To keep the simulation as accurate as possible while increasing the simulation speed, a combination of these methods can be used. The simulation can be stopped if the system is idle, this decreases the fidelity of the simulation. Furthermore, simulation steps can be combined to decrease the sample time.

Furthermore, Popovici and Mosterman describe the connection between the speed and accuracy of simulations [17]. The authors claim that finding the best trade-off between accuracy and speed is the main challenge when designing a simulation that should be executed in a given time interval.

Meftali et al. describe a multi-level SystemC simulation [76]. The authors suggest the modelling of simulation components on different levels of abstraction. This decreases the

simulation accuracy for the components described on a higher level of abstraction. This method to accelerate a simulation can only be used if the complete simulation is created anew and the components that need to be described accurately are known before the start of the development.

As Schirner et al. show in Figures 9 and 10 of their work [77], the duration of the simulation is increased while the simulation error is reduced if the level of abstraction is lowered. The authors describe the relation between level of abstraction and accuracy as follows:

> In other words, the simulation time increases exponentially with the amount of modelled features. [...] The results indicate that a loss in accuracy has to be accepted due to feature abstraction. The observed error reduces when including more features.

Further authors describe the use of Transaction-Level Modelling (TLM) instead of Register Transfer Level (RTL) modelling to increase the simulation speed [78–80]. Jung et al. also claim that the speed-up factor of TLM simulations in comparison to RTL simulations is between approx. 70 and 600.

For most XiL simulations the simulation speed is one of the main concerns. As the simulation of the virtual components needs to be as fast as the connected hardware, the accuracy of these models is often reduced. As Xia et al. state, such simulations can often be "up to 100 times slower than wall-clock-time" [81]. Xia et al. also state that one of the main challenges of their simulation is the trade-off between simulation speed and the modelling accuracy.

Work	Model Fidelity	Sample Time
Fast SystemC simulation methodology; [76]	✓	✗
Accurate RTOS Modelling and Analysis; [82]	✓	✗
Real-time Engine Modelling; [81]	✓	✗
Premature stopping and estimation; This work	✓	✓

Table 2.4: Comparison between this work and related work comparing the methods to increase the simulation speed.

In addition to these optimization techniques, this work introduces a concept that uses the inputs of a simulation to optimize the simulation speed. This input-dependent simulation speed optimization (IDSSO) is a novel concept and could not be found in the literature.

3 Design

The path to a design has a large impact on the result. Therefore, this chapter informs about the design process underlying this thesis. A large portion of this chapter deals with the co-simulation of hardware and use case, the connection of different simulation tools, and problems that occur when combining such simulators that are not intended to work together.

Another very important aspect of this thesis is the improvement of the simulation speed. This chapter also explains the used methods to improve it.

3.1 Expanding Model-in-the-Loop Simulations

In classic MiL simulations the models of the hardware and the environment are crude. This kind of simulation is used to verify that a certain function would solve the problem at hand. After this step more advanced models are created and other simulation types (SiL, PiL, HiL) are used to verify the functionality.

Currently, there is no concept for a simulation of an advanced system simulation in conjunction with the environment (henceforth System-Simulation-in-the-Loop (SSiL)). Such simulation could be used to verify that the complete system behaves as intended before the hardware is being built. The following sections describe the design we created in this thesis to construct an SSiL simulation.

3.1.1 Requirements of the SSiL simulation

The functional requirements of an SSiL simulation do not differ much from the requirements of any other XiL simulation. These requirements are:

- Stimuli that arise from the use case need to be generated.
- The stimuli need to be transmitted between the simulators.
- The simulators need to be synchronized.

In the case of an SSiL simulation, the requirement for the synchronization is more strict. The simulators need to be synchronized after every simulation step. This comes from the fact that the complex behaviour of a system allows the spontaneous creation of messages

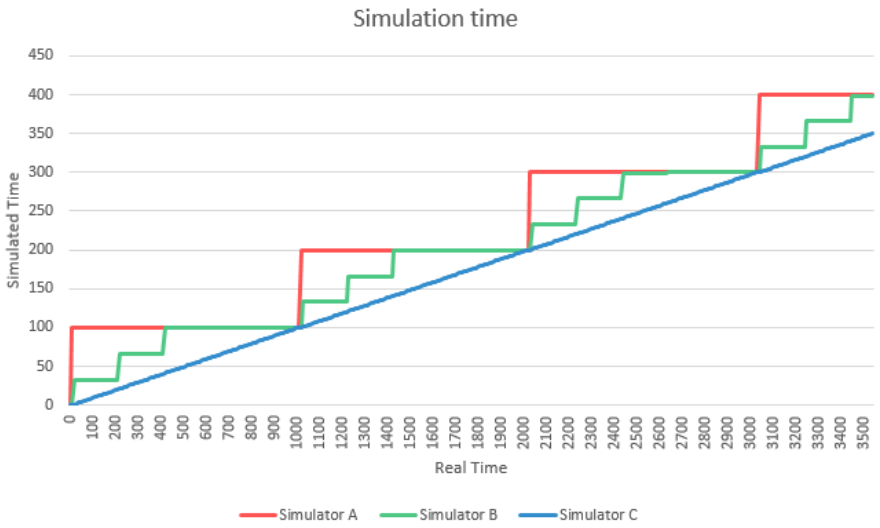

Figure 3.1: Time difference of different simulators when synchronized after every time step.

that need to be captured in the other simulation. Therefore, the synchronization needs to be kept as tight as possible.

In contrast to the lock-step synchronization shown in Section 2.2.2 Figure 2.1, Figure 3.1 shows the synchronization of multiple parallel simulations. In the case shown in Figure 3.1, three simulators using different time steps are used. As the simulations are executed in parallel, and different simulators may use different time steps, the simulators can only be synchronized at time instances of the least common multiple of all time steps. At this time, all simulators need to wait for the slowest one to finish the calculation until this time step. In Figure 3.1 simulators A, B, and C have 100, 33.3, and 1 simulation time ticks respectively. Furthermore, Simulator A finishes with its calculation step the fastest while Simulator C is the slowest to have reached 100 simulation time steps. Therefore, Simulators A and B need to wait until Simulator C has reached the common synchronization time step.

Following this, the simulators in this thesis will use time steps that allow a synchronization as often as possible. This means that the simulator with the largest time step should be able to synchronize to all others in every time step. All other simulators that are connected will therefore use time steps which fit N times into the larger interval ($N \in \mathbb{N}$).

3.1.2 Stimuli Generation in XiL Simulations

In MiL simulations the stimuli are usually generated by a simplified model of the environment of the hardware under development. These environment models are tailored to fit the required stimuli. As this approach is typically static, a new model of the environment needs to be created for each use case tested.
As the model and the environment are simulated on a functional level, both parts of the simulation can be created using one simulation environment. As an example Brockmeyer et al. used Matlab/Simulink to create the functional model and the model of the

environment to perform the MiL simulation of their system [16].

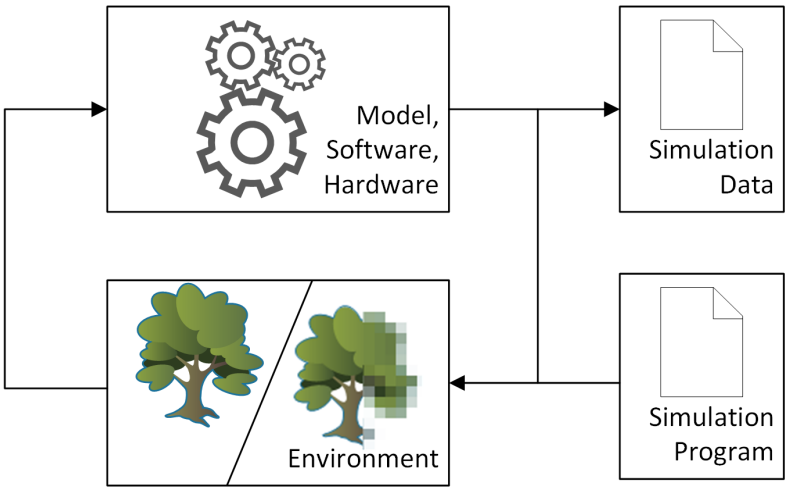

Figure 3.2: Basic concept of an XiL simulation. The model gets stimulated by the environment and the environment can in turn be changed by the model.

Any XiL simulation follows the overall model shown in Figure 3.2. The two components of the simulation are the model of the hardware and the model of the environment. The environment stimulates the model and reacts to the output of the model. For most MiL simulations the model is very abstract and does only represent the functionality of the desired hardware, the environment is also modelled very abstractly. Due to this abstraction, the simulation of the environment can be performed quickly, but poses restrictions on the capability of the environment simulation.

When using a more complex simulation of the environment, a wider array of use cases can be simulated with one model. Furthermore, a more complex interaction between the environment and the hardware under development can be implemented and tested.

For many MiL simulations, the model of the environment and the hardware are created in the same simulation environment such as Matlab/Simulink [16]. For following XiL simulations (SiL, PiL, HiL, ViL) the model of the hardware is then replaced.

The design of the SSiL simulation in this thesis follows the same approach. We replace the model of the hardware of the XiL simulation by a more complex model that is capable of executing a program and reacting to events in the environment. Furthermore, the simulated environment is also be replaced by a more complex environment simulation allowing more complex and realistic scenarios.

The model complexity directly influences the accuracy and the speed of the simulation. A more complex model typically yields more accurate results but is slower to compute [25].

3.1.3 X-in-the-Loop Co-Simulation

When using different simulators to simulate the model of the hardware and the environment the simulation becomes a co-simulation [38, 39]. Due to the simulation of the

components using specialized simulation tools, more details can be seen in the simulations and more accurate results can be created. This increase in detail comes at the cost of modelling the components in separate simulation tools and connecting the simulators. Furthermore, the increase in detail implies that more calculations are needed and therefore the overall simulation becomes slower.

In the co-simulation of this thesis, we use two simulators: SystemC to perform the simulation of the hardware, and Gazebo to simulate the environment.

3.1.4 Synchronizing the Simulations

A very important part of any co-simulation is the synchronization between the simulation tools.

In the case of an XiL simulation, both simulators can send information to the other one at any simulation time. Thus, the simulators need to be fully synchronized. Wu et al. showed that using a lock-step synchronization, all components of a co-simulation are synchronized at the end of every time step [54]. When using simulators that use different time step durations, the synchronization can be achieved at the least common multiple of the time steps. This is shown in Figure 3.3.
Also the work in this thesis uses a lock-step approach to keep the simulators synchronized.

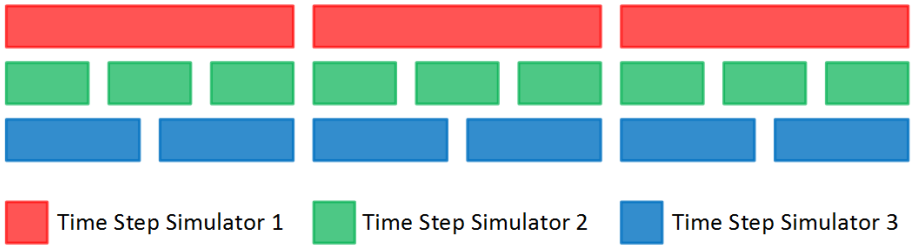

Figure 3.3: Time steps of multiple simulators and synchronization points.

3.2 Optimizing the Simulation Speed of Co-Simulations

Next to the accuracy of simulations, another core metric is the simulation speed. These two parameters are linked to each other. If more accurate results are needed, the simulation will perform more calculations and thus the simulation will be slower (Section 2.2.7).

If the simulation allows interactive inputs from the operator, the simulation speed may need to be optimized after the model is created. At this time the model fidelity, sample time, and number of solver iterations are fixed.

To solve the issue of optimizing the simulation speed after the simulation model has been created, we use the inputs of the simulation to optimize the simulation speed.

The basic concept of the Input Dependent Simulation Speed Optimization (IDSSO) is to estimate the effects a certain input has on the simulation variables at the end of the time step. If the effects can be estimated, the simulation step is skipped and the estimations are used as result.

This technique of using estimations introduces errors in the simulation. To minimize these errors the optimization should not be used during the whole simulation. Furthermore, when using estimations to calculate parameters used further, the errors accumulate and lead to large errors over time.

3.2.1 Simulation State Estimation

If the simulation runs too slow to be useful, but the model cannot be changed to a less accurate but faster model, estimation of simulation states is a tool that we examined to increase the simulation speed. This effectively increases the simulation speed at the cost of accuracy.
The question whether it is useful to implement an estimation process for an already existing simulation cannot be generally answered for all use cases.

The estimation of simulation states is not always possible, therefore such methods increase the jitter on the simulation times. Figure 3.4 shows an example of this. The calculations for each time step take different amounts of time. This can lead to varying overall simulation speeds. To estimate a general speed improvement, we designed micro-benchmarks to estimate the speed increase for the most common states the simulation is in. With an analysis of the amount of time spent in each of these states, an overall speed improvement is estimated.

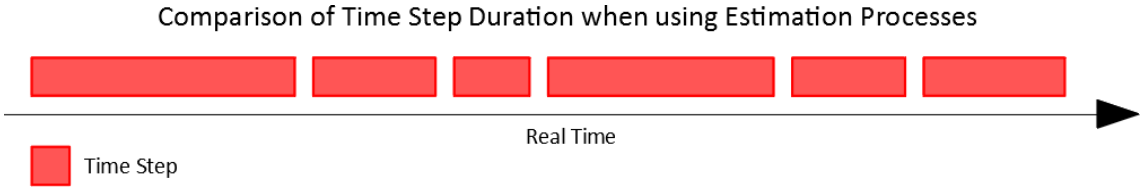

Figure 3.4: Real time used for steps when using estimation techniques.

Linear State Estimation

Simulations essentially take the current simulation state and other inputs and generate a new state from it. The calculation of the inner states is described by the model. In normal simulations we are interested in the inner states of the simulation. Thus, a simulation can be seen as a non-linear function that takes three parameters, the model, the current simulation state and other inputs, and generates a new simulation state (Equation 3.1).

$$State(t+1) = Simulation(Model, State(t), Inputs(t)) \tag{3.1}$$

In XiL simulations the other input is generated by the (simulated) environment. In addition to this, the XiL simulation can produce outputs that affect the environment. This can be described by Equation 3.2.

$$\begin{aligned} State_S(t+1), Outputs_S(t+1) &= Simulation(Model_S, State_S(t), Outputs_E(t)); \\ State_E(t+1), Outputs_E(t+1) &= Environment(Model_E, State_E(t), Outputs_S(t)) \end{aligned} \tag{3.2}$$

In XiL simulations, the calculation of the *Simulation* function is considered as the bottleneck for the simulation speed. Thus, to increase the simulation speed, optimizations for these functions are needed.

To estimate the change of the simulation state $State_S$, the derivative of the function needs to be calculated. This requires the $Output_E$ input to the function to be constant.

The derivative of any function $f(x)$ at $x = t$ is defined as:

$$f'(t) = \lim_{h \to 0} \frac{f(t+h) - f(t)}{h} \tag{3.3}$$

Thus, the derivative of *State(t)* of Equation 3.1 can be approximated as

$$State'(t) = \frac{State(t) - State(t - \Delta t)}{\Delta t} \tag{3.4}$$

From this the state at the next discrete time interval can be approximated as:

$$State(t + \Delta t) = State(t) + State'(t) \cdot \Delta t \tag{3.5}$$

To apply this logic to the *Simulation* function in Equation 3.2, $Outputs_S(t+1)$ also needs to be derived. This leads to

$$\begin{aligned} State'_S(t) &= \frac{State_S(t) - State_S(t - \Delta t)}{\Delta t}; \\ Outputs'_S(t) &= \frac{Outputs_S(t) - Outputs_S(t - \Delta t)}{\Delta t} \end{aligned} \tag{3.6}$$

Using these equations it is possible to estimate the next simulation state $State_S(t + \Delta t)$ and the output $Output_S(t + \Delta t)$ under the prerequisite that the input $Output_E(t)$ does not change.

As a simulation might also trigger internal state changes based on internal states or timers, these need to be modelled as additional inputs. Therefore, this optimization is only possible if the internal workings of the model are well known and these triggering states are accessible.

Furthermore, if the inputs $Outputs_E$ change, the simulation needs to be executed to generate the needed states ($State_S(t)$ and $State_S(t-1)$) to perform an estimation.

As $Output_E$ needs to be constant for the estimation to be applicable, such estimation increases the simulation speed while the simulation is idle.

Based on the model and time step, it can be possible that such an estimation is not

accurate enough. Then, further simulation steps need to be calculated to gain enough data for a better estimation. For example, if the data of three simulation steps is used to also calculate the second discrete derivative of the state, the estimated state does not change linearly in time but quadratically.

For the example simulation in this thesis we use the first derivative of the states to calculate the state change.

Applicability of the Estimation

The estimation in Equation 3.6 is correct for static inputs $State_S(t)$, $Outputs_E(t)$ and $\Delta t \to 0$. As Δt is not 0 for event-driven simulations, and $State_S(t + \Delta t)$ may differ from $State_S(t)$, an error is introduced by every step taken using the estimation.

Using the estimation to calculate a new $State_S(t+\Delta t)$ effectively advances the simulation time by Δt. The effects of this are explained in Section 3.2.2.

The input to the simulation $Output_E(t + \Delta t)$ is itself calculated by another simulation. The difference between two such inputs may not be trivial to calculate. Therefore, the effects of such change are also not trivial to calculate. Thus, the estimations in Equation 3.6 are not applicable if $Output_E(t + \Delta t)$ is different from $Output_E(t)$.

3.2.2 State Changes due to the Estimation

The above estimations linearize the model at the given state $State_S(\tau)$. This introduces errors if the current state of the simulation $State_S(t)$ differs from $State_S(\tau)$. To keep the error small, the difference Δ between these values needs to be small. The calculation of the difference of such simulation states may not be trivial. Therefore, it is required that $State_S(t)$ differs minimally to $State_S(t + 1)$. Therefore, the difference between states can be estimated by the time between these states. This leads to the conclusion, that after some time T the difference Δ is too big, and the linear estimation introduces too much error. At this time $\tau + T$ the simulation needs to be executed anew to linearize the model at $State_S(\tau + T)$.

It is furthermore possible to use higher-order equations to calculate the estimations. This may increase the usefulness of the estimation in such way that T can be chosen differently.

Figure 3.5 shows a simulation of the charging and discharging of a capacitor. In this figure the ground truth (*Ref*) is calculated. The unoptimized simulation (*Sim*) matches the ground truth. Additionally four estimations are shown in this figure. Two that use a linear model (*Lin50*, *Lin100*) and two that use a quadratic model (*Qua100*, *Qua200*) to estimate the charge on the capacitor. In this example T is chosen to be 50 and 100 for the linear estimations and 100 and 200 for the quadratic ones. After T steps are calculated using the estimation, the simulation is used for 2 and 3 steps respectively to calculate the new parameters for the estimation.

A statistical analysis of the errors that arise in the simulation and the estimates are plotted in Figure 3.6. From these errors one can see that, for the model of a capacitor, a quadratic estimation can reduce the introduced error at least as good as a linear estimation using a T that is halved. But, when doubling T for the quadratic estimation, the

introduced error is bigger than the error in the linear case.

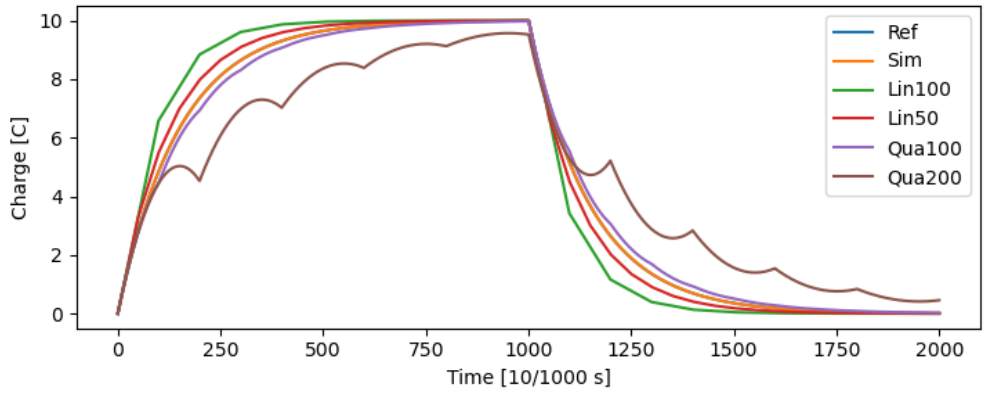

Figure 3.5: Simulation of charges at a capacitor. Calculated and estimated.

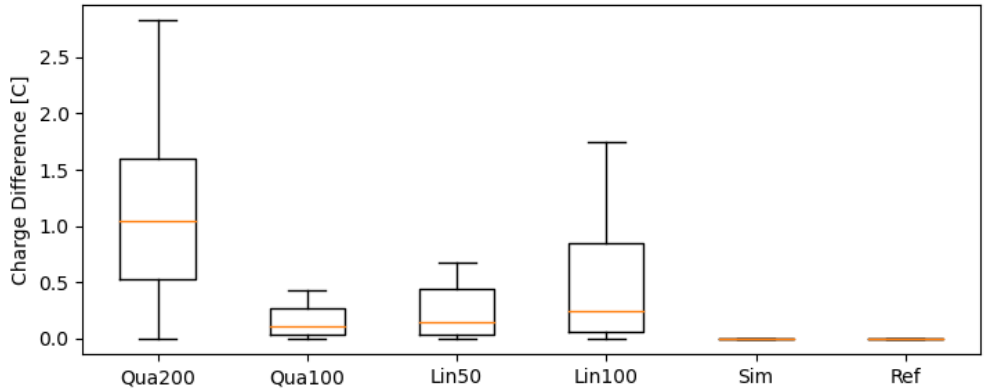

Figure 3.6: Errors of the simulation and the estimations.

3.2.3 Applying the Optimizations to SystemC

As SystemC does not natively support the skipping of time steps and using an estimation instead, we developed a method that allows us to do this.

This method makes use of adding data to the generated trace file that is used in a post-processing step. The data added signals the post-processing step that some simulation time has been skipped. In the post-processing step the datapoints after the signal are delayed by the amount of skipped time. After inserting the signals to the post-processing, the estimated internal states are updated. When the simulated hardware is idle multiple such signals are created in a short amount of time. This effectively compresses the trace of the simulated time while the hardware is idle. During post-processing the skipped time is added back.

4 Implementation

The following chapter sheds light on the most important implementation aspects. It provides insights into the combination of the two simulators that simulate the environment and the hardware system, respectively. This chapter also describes the used optimization strategies. Furthermore, this chapter describes a hardware prototype of a smart sensor system that serves as a test case for the designed SSIL co-simulation.

The SSIL co-simulation uses a simulation of the environment of a hardware system to drive the simulation of the hardware. Thus, the hardware simulation is integrated in the simulation of the environment. Figure 4.1 shows a concept on how to integrate a SystemC simulation with an environment simulation. Here the environment simulation calculates stimuli for the SystemC simulation and transmits them to the SystemC simulation. To complete the simulation step, the environment simulation waits until the SystemC simulation has finished.

This process effectively creates stimuli for a hardware simulation based on a high-level description of the intended use case. This speeds up the stimuli generation compared to the current practice of an engineer reverse engineering the use case and extracting the stimuli manually from that. In contrast to other methods of deriving stimuli for a hardware simulation (e.g., [83–85]), this method is deterministic and does not rely on machine learning algorithms.

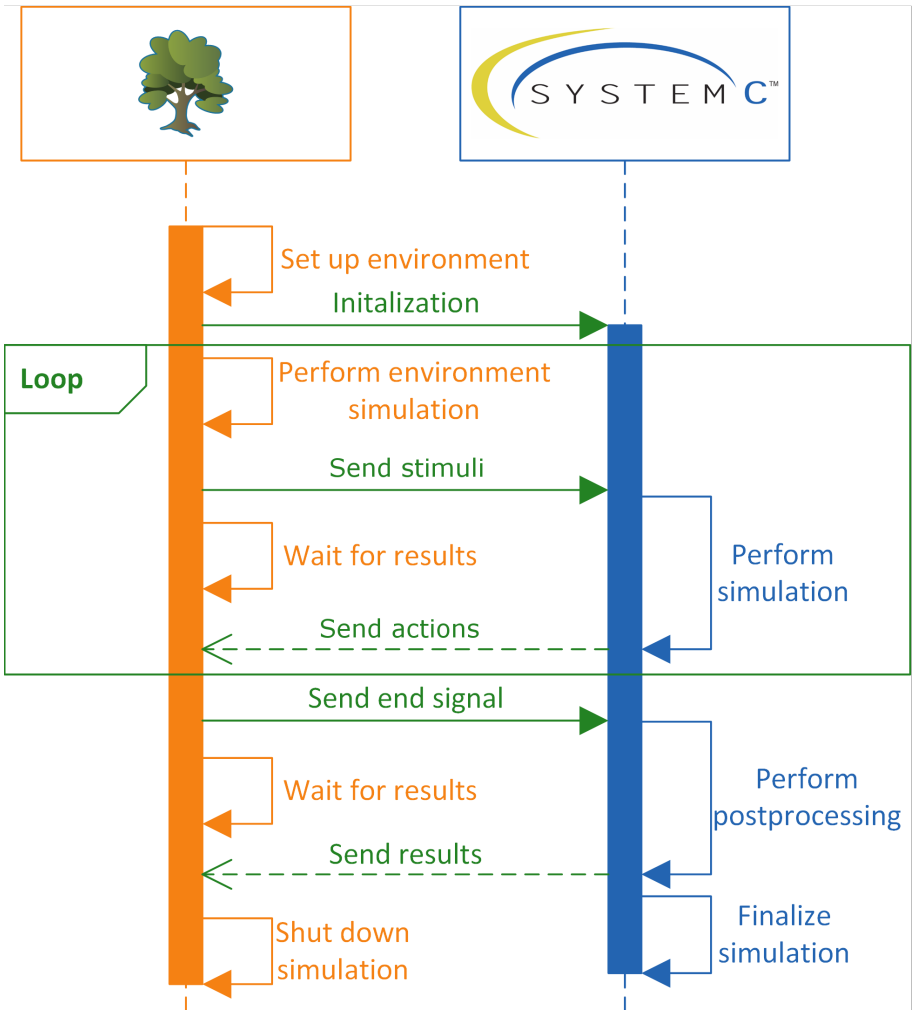

Figure 4.1: Concept for integrating the SystemC hardware simulation with an environment simulation to generate Stimuli.

4.1 Requirements for the implementation

There are two requirements for the implementation of SSiL simulations, which concern the simulation speed and accuracy (Section 3.1.1).

- The final co-simulation should not be significantly slower than the hardware simulation on its own.
- The final accuracy of the co-simulation should be comparable to the accuracy of the hardware simulation alone.

These two requirements can also be quantified and be used to determine the quality of the resulting co-simulation.

From these high-level requirements, demands on the interface between the simulations can

be inferred. These demands for the interface can be split according to the simulators and expressed as needed steps.

In the proof-of-concept co-simulation, the Gazebo simulator is used to simulate the environment. Therefore, the needed steps are tailored to the Gazebo simulator.

RG.1 Collect all necessary data from the environment. This is sensor data, communication data, and simulation status information.

RG.2 Package the collected information and transmit it.

RG.3 Until the SystemC simulation is finished, halt the environment simulation.

RG.4 While waiting, collect all received information sent by the SystemC simulation and sort it by the intended recipient.

RG.5 When the SystemC simulation step is finished, forward the collected information to the recipients of the simulation.

The implementation of these steps is done in a separate Gazebo plugin that acts as the interface to the hardware simulation. Figure 4.2 shows a possible order of these steps.

To be able to communicate with the environment simulation, the hardware simulation (SystemC in this case) needs to implement a matching interface and support the following steps.

RS.1 All data intended for the simulation needs to be received. The simulation must be halted until all data is received.

RS.2 The SystemC simulation needs to be halted at the simulation time of every time step of the environment simulation. Furthermore, the SystemC simulation needs to be restarted at these points.

RS.3 Unpack the received information and provide it to the SystemC simulation in a usable form.

RS.4 When resuming the simulation, match the time step to the time step of the environment simulation to keep the simulations synchronized.

RS.5 At the end of the time step, SystemC and Gazebo must be synchronized.

RS.6 As the SystemC simulation is slower than Gazebo, measures to increase the simulation speed should be taken when possible.

The interface to the Gazebo simulation acts as the testbed for the SystemC simulation. Figure 4.3 illustrates how the interface can be implemented on SystemC side.

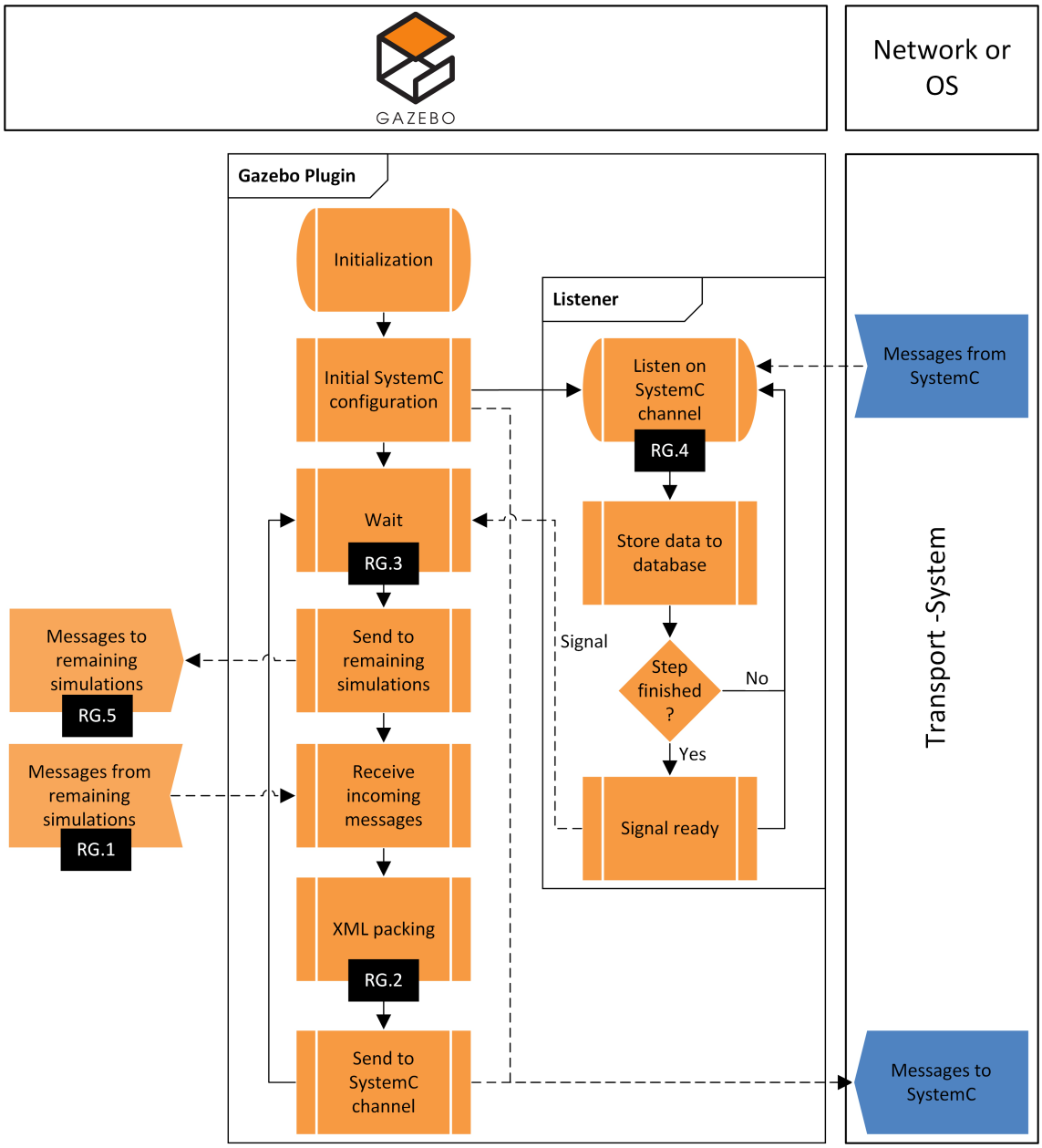

Figure 4.2: States the communication interface on the Gazebo side needs to have.

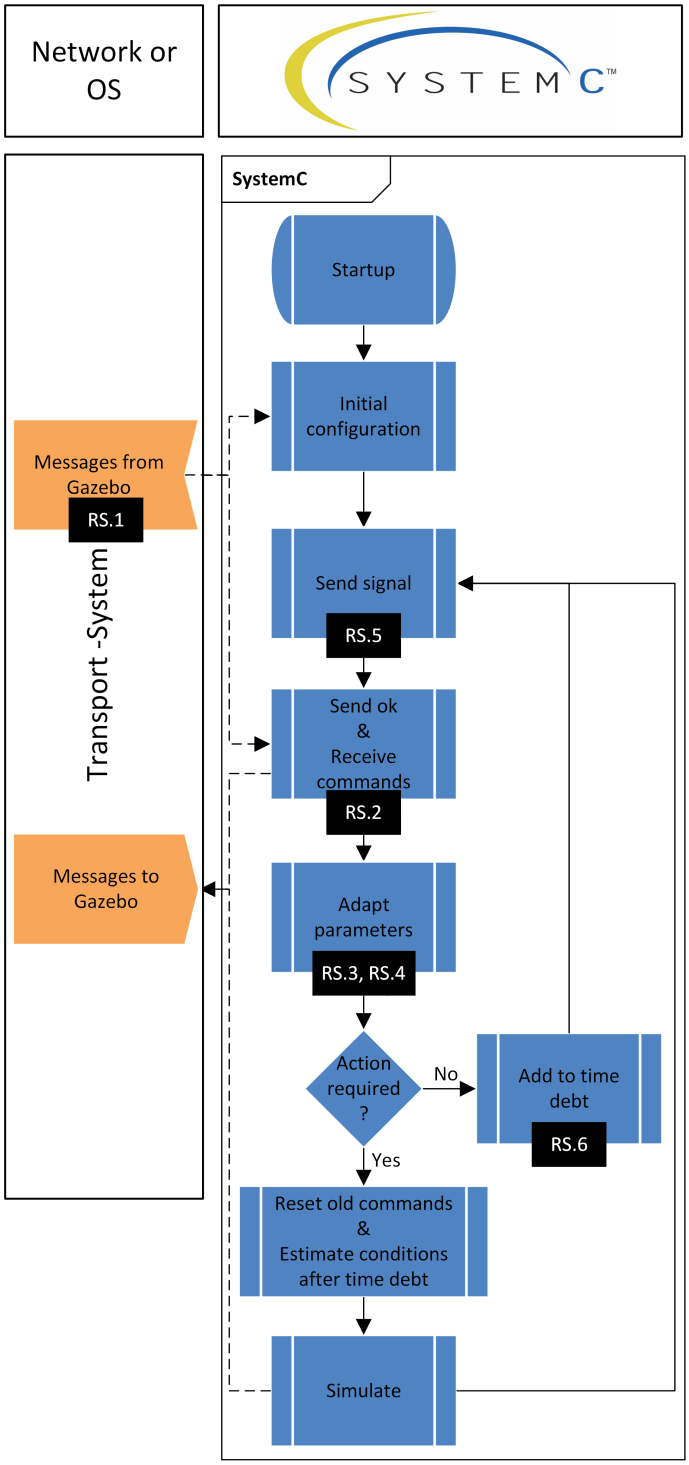

Figure 4.3: States the communication interface on the SystemC side needs to have.

4.2 Connecting Unlike Simulators

In Section 3.1.2 it is described that in XiL simulations the model gets stimulated by the environment. As the environment is also simulated in the SSiL simulation two simulators are connected to form a co-simulation. The two simulators can be connected in different ways. These possibilities come with different challenges and advantages, which we describe below.

4.2.1 Instantiation

There are two ways to instantiate the SystemC simulations and connect them to the Gazebo simulator.

The first one is to instantiate them as part of the setup routine in the Gazebo plugin. This is done by forking from the current process and executing the SystemC simulation as a child process.
By doing so the new process can be connected to the Gazebo simulator via POSIX-pipes. These pipes can transport text between the simulators. This way of instantiating the SystemC process entails that it is executed on the same computer as the Gazebo simulation. Figure 4.4 shows the sequence that is needed to start the SystemC simulation from the corresponding plugin. During the initialization of the plugin POSIX pipes are created. When the *fork* syscall is executed, the child process connects the pipes to the standard in- and output. Thereafter the SystemC process is executed. The parent process still holds handles to the pipes. This realizes the communication channel between the Gazebo plugin and the SystemC process. To be able to receive the data from SystemC whenever some is available, a thread that listens on the receiving pipe is created. The functions *onUpdate* and *onEndSignal* are called by the Gazebo controller. These two functions generate messages that are sent via the outgoing pipe to the SystemC simulation. These messages are received in the *sc_main* function. There the messages are parsed and the SystemC simulation is handled accordingly. During the execution of the SystemC simulation, data may be sent back to the Gazebo simulator. There the listening thread receives the data and stores it to be used by the simulation.

Method two of instantiating the SystemC simulation does not have the limitation that the SystemC simulation needs to be executed on the same computer as the Gazebo simulation. Using this method all components of the simulation are started separately. To connect to the Gazebo simulation, each SystemC simulation requires a configuration that lets it find the Gazebo simulation. The SystemC simulation can then connect to the Gazebo simulator via the network. The Gazebo simulator at the other side needs to be able to wire the incoming connection to the corresponding plugin. Figure 4.5 shows the sequence of actions that are needed to establish such connection via the network.

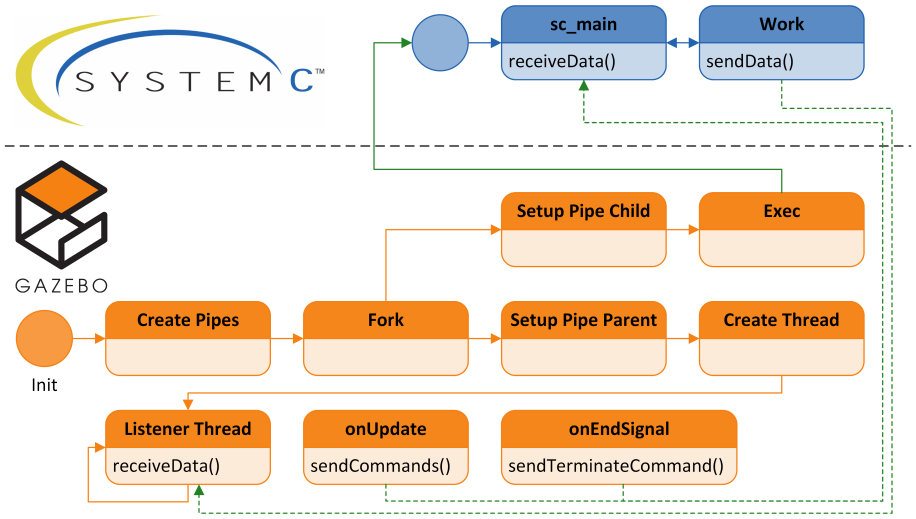

Figure 4.4: Sequence of actions to start a SystemC simulation by forking from the plugin.

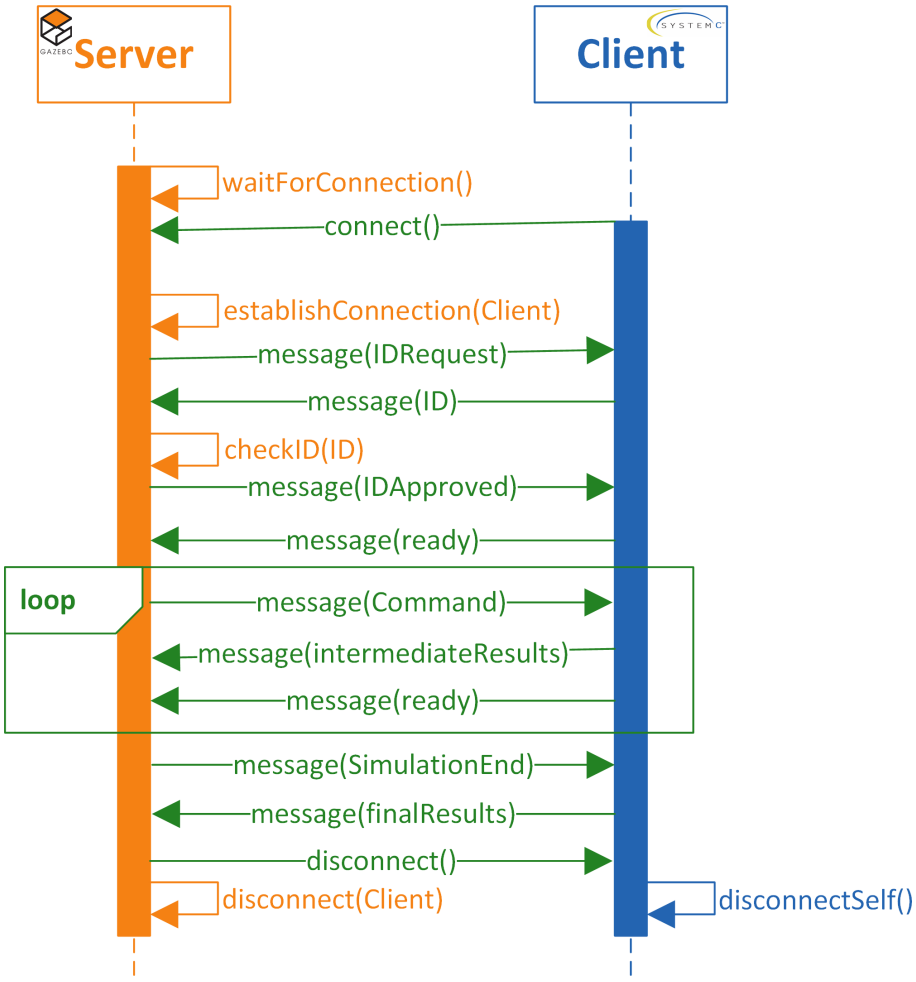

Figure 4.5: Sequence of actions to connect a SystemC simulation to Gazebo via the network.

4.2.2 Communication Between the Simulations

Both solutions to instantiating the SystemC simulations and connecting them to the Gazebo simulator support the transmission of byte streams. Therefore, a text-based serialization scheme, such as the Extensible Markup Language (XML), the Java Script Object Notation (JSON), or YAML (YAML Ain't Markup Language), can be used.
As Kazuaki Maeda stated: "*XML is currently used as a standard language for data representations in [a] wide application area.*" [86]. Therefore, XML is being used to serialize the data that is communicated between the simulations.

4.2.3 Timing Differences

A major problem that can be noticed is the difference in the time resolution between the simulation steps. While Gazebo operates in steps of 1 ms, SystemC can be operated using time steps of 1 fs. This means that SystemC usually takes more time to simulate a given duration. This difference needs to be accounted for when implementing the interface between the simulators.

In order to operate correctly, special care needs to be taken of the synchronization between the simulations. When sending the data for the SystemC simulation without waiting for a SystemC response that the simulation step is finished, unexpected behaviour, such as long communication delay and huge buffer sizes, can occur. Such behaviour is not faulty per se, but can lead to the false conclusion that the SystemC simulation is faulty. Therefore, a proper synchronization protocol where each partner waits for the other one needs to be implemented. Figure 4.6 illustrates the communication that is needed to be synchronized at the end of each Gazebo time step. When using this protocol the simulators operate in a lock-step manner (Section 3.1.4). Therefore, SystemC only receives the data it needs for the current time step and notifies Gazebo when the time step is finished. This notification at the end of every time step also signals that the simulation is operational. This also avoids overly large buffer sizes.

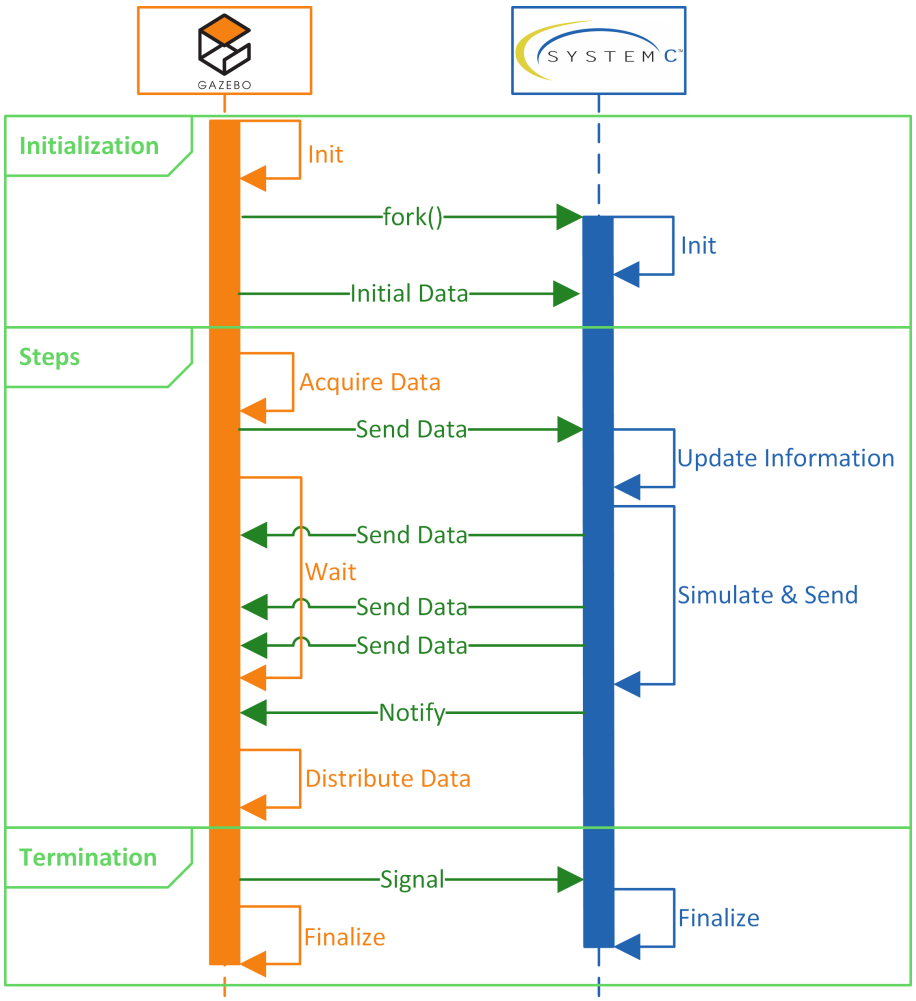

Figure 4.6: Communication to be synchronized at the end of a Gazebo time step.

4.2.4 Speed Bottlenecks and Remedies

SystemC is normally used to test the functionality of hardware, but not the effects that a longer runtime has. Therefore, SystemC simulations can be constructed to perform cycle-accurate simulations. This accuracy comes at the cost of a longer simulation time.

The Gazebo simulator on the other hand is used to simulate the behaviour of robots. These simulations tend to include idle times and usually simulate longer durations. The simulation itself calls every component once every time step. This behaviour is used to have a high simulation speed. To be accurate, Gazebo is implemented single threaded. This entails that a single low-performance module causes the whole system to be slow.

This simple analysis shows the major bottlenecks of a combination of the two simulators:

1. A SystemC simulation is slower than the Gazebo simulation. This slows the overall simulation.

2. The single-threadedness of Gazebo prevents a partial simulation of the next simula-

tion step while the SystemC simulation is active.

3. The single-threadedness of Gazebo furthermore prevents multiple SystemC simulations to run in parallel.

To minimize the impact of a slow SystemC simulation on the overall simulation (Bottleneck 1), the accuracy of the SystemC simulation is reduced by estimating the results while the simulated hardware is idle. This estimation is described in Section 3.2.

Bottlenecks 2 and 3 can be remedied by interleaving the simulation steps of the simulators. The change that needs to be done in order to allow that, is that the interface module in the Gazebo simulation needs to wait for the completion of the SystemC simulation at the beginning of its turn rather than at its end. Figure 4.7 shows the activation of the simulations when implementing the non-interleaved or interleaved interface. The non-interleaved simulation behaves in a single-threaded manner. In the interleaved version, a part of the Gazebo simulation, as well as the SystemC simulation can be active at the same time. The communication for the interleaved simulation is depicted in Figure 4.8. This figure reveals that the results of the SystemC simulation can only be received in the next Gazebo time step. This results in a delay of one Gazebo time step in the communication. This interleaving of simulation steps can only increase the simulation speed until all cores of the CPU are used. To keep increasing the parallelism of the computation, the SystemC simulations can be outsourced over a network. Further details on the parallelization of the SystemC and Gazebo components can be found in Section 4.3.

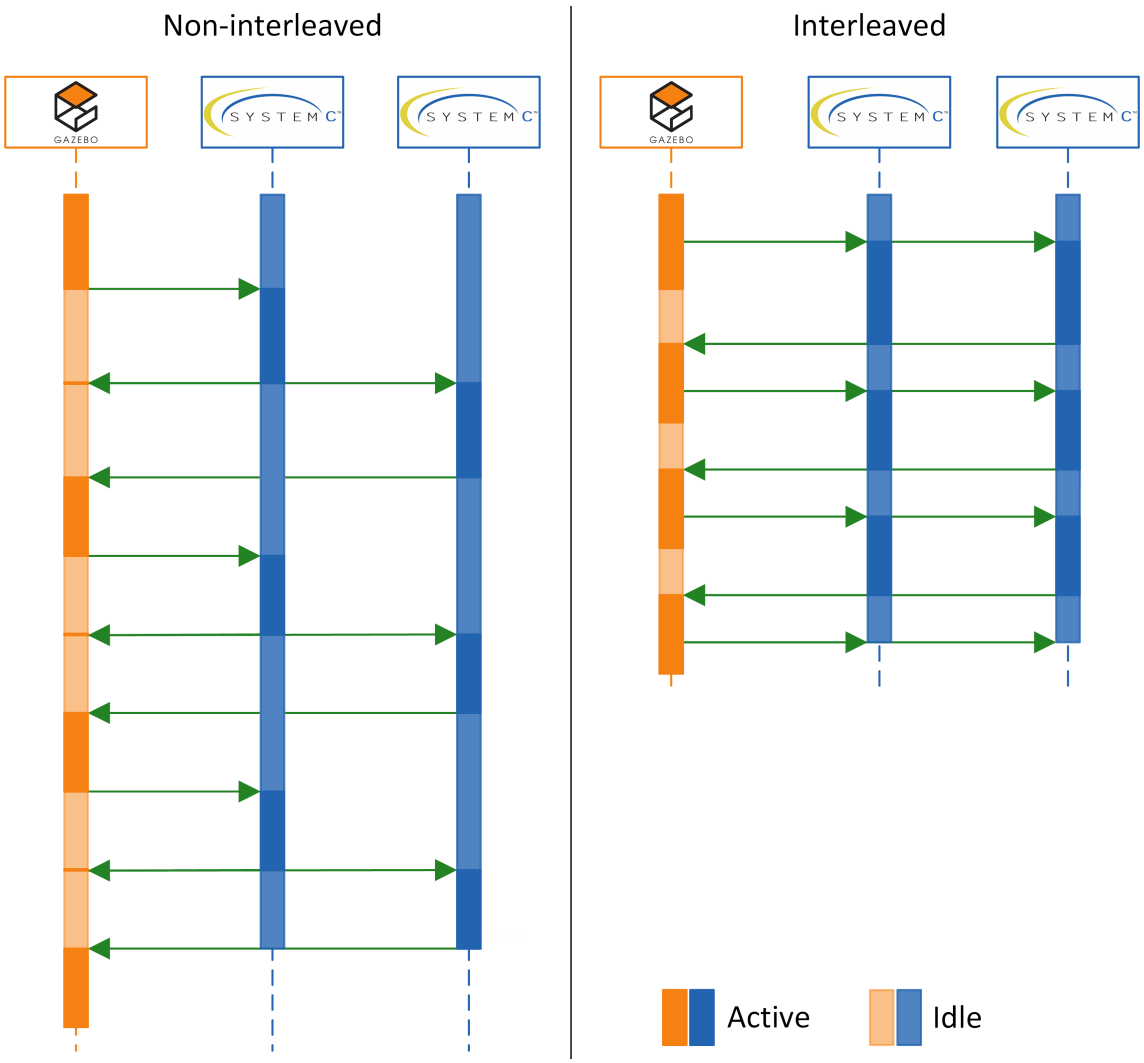

Figure 4.7: Comparison between the execution of non-interleaved and interleaved simulations.

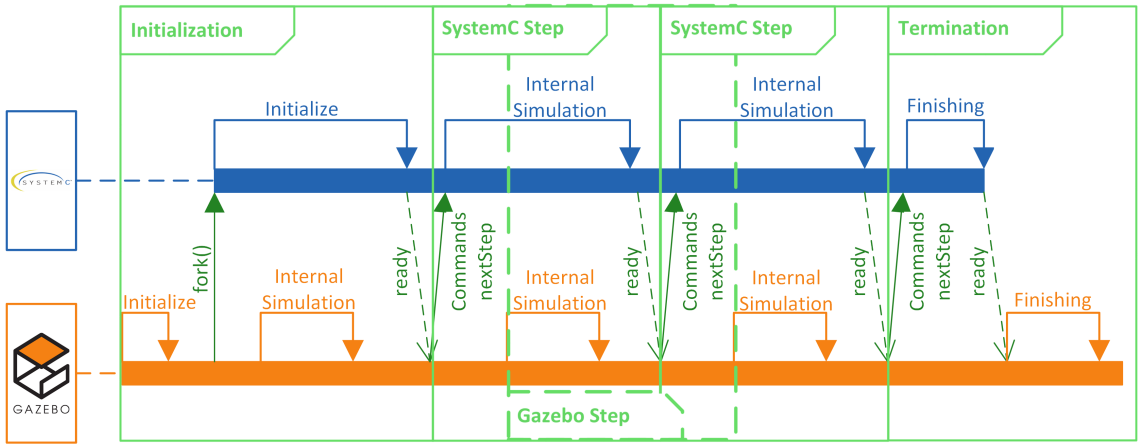

Figure 4.8: Sequence diagram for interleaved simulations.

4.3 Parallelizing the Simulations

When one wants to simulate a network, multiple sensor nodes need to be placed in the environment. As Gazebo is blocked by the execution of a sensor plugin, the simulation of multiple sensor nodes is done sequentially. To circumvent this, two approaches are possible. These are described in the following sections.

4.3.1 Parallelization by Interleaving Simulation Steps

When interleaving the simulation steps, the code for the sensor plugin only requires little change. Instead of waiting for the end-signal of SystemC at the end of the plugin update routine, the wait-step is performed at the beginning. This simple measure allows the parallel execution of SystemC processes if CPU cores are available. One core is needed to perform the Gazebo step. The rest can be used to perform the SystemC simulation. Figure 4.9 highlights the differences between the non-parallelized and interleaved simulation. Here the *Steps* portion of the sequence diagram has been cut out and compared to each other.

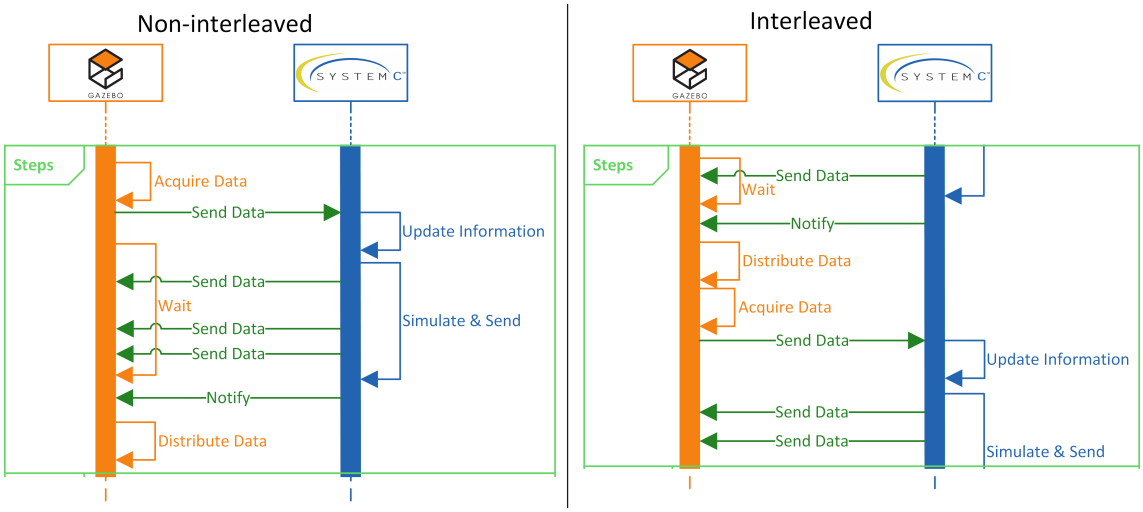

Figure 4.9: Difference between the non-parallel and the interleaved simulation.

4.3.2 Parallelization by Using a Central Connection

If even more nodes need to be simulated, the SystemC simulations need to be outsourced to different computers. Therefore, the simulations cannot be forked directly from the sensor plugins. In this scenario the SystemC simulations need to be started on another computer and connected to the Gazebo simulator. This forms a star topology where each SystemC simulation is directly connected to the Gazebo simulation. To handle the incoming requests, a server plugin needs to be created. This server can then create a communication thread for each connection. These threads need to determine to which sensor plugin the SystemC simulation needs to be connected. The sensor plugin is not required to handle the communication directly. The data they need to transmit to their

SystemC simulation is handed over to the server plugin. The communication thread for this simulation passes on this information. The communication thread now listens for incoming packets from their SystemC simulation and stores the data. If the end-signal is received, the communication thread forwards the data to the sensor plugin. If all communication threads have received the end-signal, the simulation can proceed.

In addition to this parallelization, the interleaving of the simulation steps can also be performed.

Figure 4.10 shows the communication that is performed for each sensor plugin. The sensor plugin first collects the required data. This information is sent to the communication thread of the server plugin. The communication thread appends information about the simulation status. This information is then packed and sent via a network to the receiving computer. There the SystemC simulation receives the packet and processes it. If data is generated during this simulation step, the data is transmitted to the computer running the Gazebo simulation. At the end of the simulation step a signal is generated that notifies the Gazebo simulation that the step has been executed. The communication thread receives this signal and forwards the information it received in the meantime to the sensor plugin.

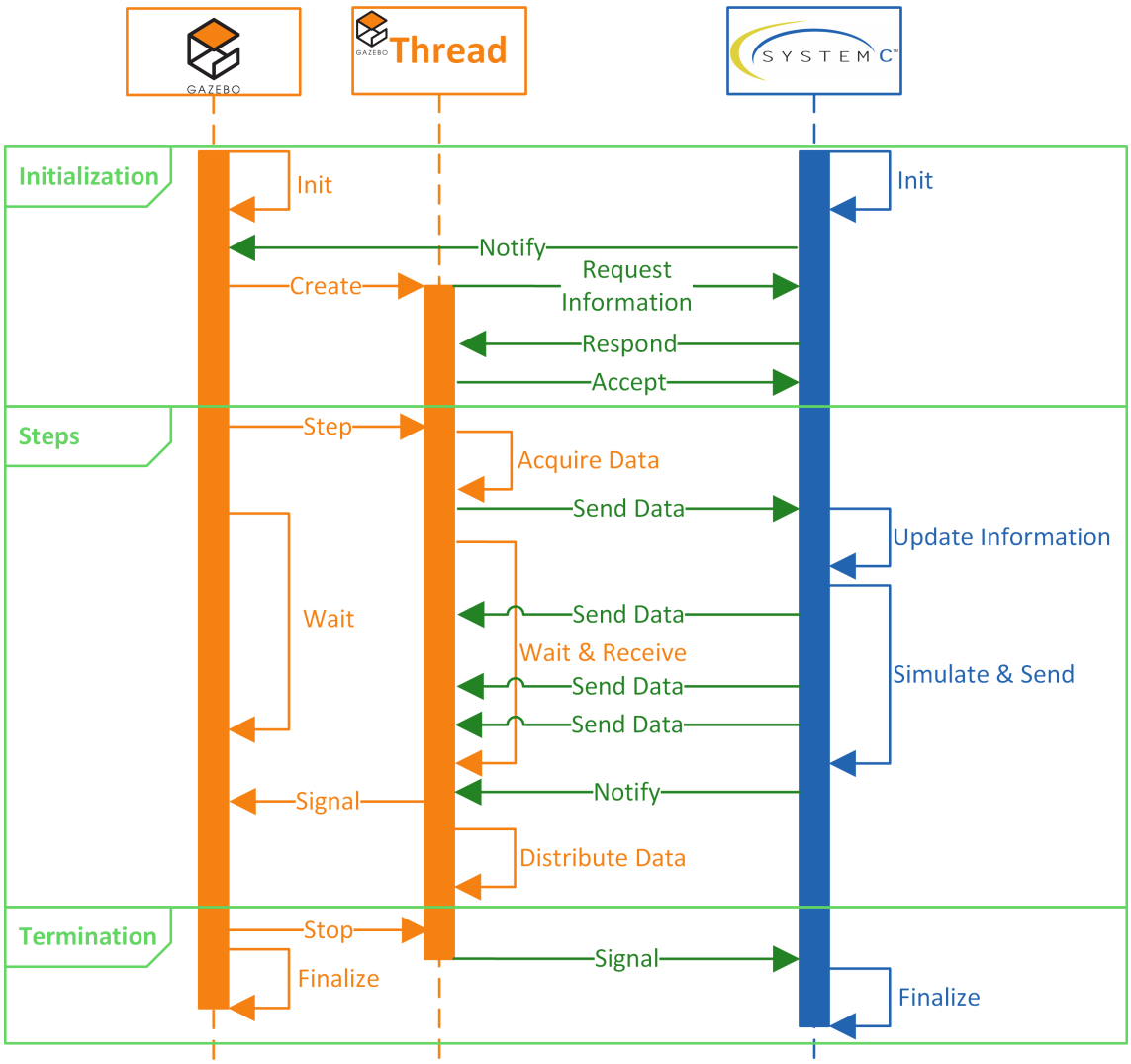

Figure 4.10: Communication that is performed to use SystemC simulations on different computers.

4.4 Example Simulation

To test the simulator an example simulation needs to be developed. During the IoSense project, a simulation of a sensor system was needed. This simulation was used to test the constructed simulator and evaluate it. In this simulation, a complete smart sensor was modelled. This sensor consists of a micro controller (uC or μC), a flash memory, an additional security controller (SC), an NFC controller that is capable to harvest energy, and two ports to external sensor hardware. A block diagram of this smart sensor is shown in Figure 4.11.

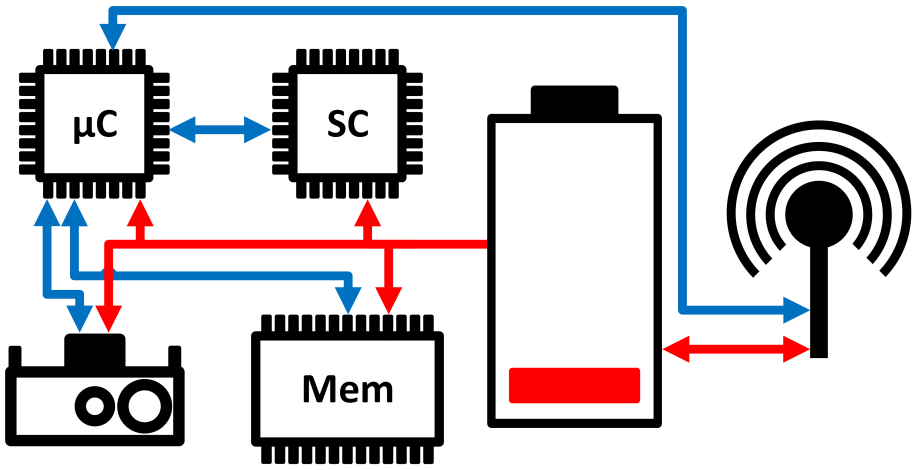

Figure 4.11: Block Diagram of the simulated smart sensor.

4.4.1 Acquiring Data for the Simulation

To get exact figures for the test simulation, two research prototypes have been designed. The first prototype (Prototype A), shown in Figure 4.13, is used to test the capabilities of energy harvesting from a Near Field Communication (NFC) field and to test the possibility of operating a sensor from a super-capacitor. This prototype consists of an MSP430FR5969 evaluation board that is expanded by a custom-built Printed Circuit Board (PCB). This PCB connects the energy harvesting interface of the NFC controller to the super-capacitor of the evaluation board. Via this connection the super-capacitor can be charged and the sensor operated. Figure 4.12 shows the wiring diagram to connect the NFC controller to the super capacitor of the evaluation board. Additionally, the PCB connects the data lines of the NFC controller to General Purpose Input Output (GPIO) pins of the evaluation board. This research prototype has then been charged and discharged to measure the capabilities of this energy provisioning system.

The second research prototype (Prototype B) is used to evaluate the capabilities of the modelled smart sensor. A photo of this prototype can be seen in Figure 4.14. This prototype consists of two PCBs. The first one is the sensor node itself. This sensor node prototype has a μC, a flash memory, an NFC interface, an SC, and two external ports for sensor hardware. To supply these elements with energy and to measure each element separately, a separate power line for each element is linked to an external port. To reduce the energy consumption of the complete sensor node even further, each power line can be interrupted with a power switch. The second PCB of Prototype B holds the power supply. This power supply can regulate the input voltage for the smart sensor. Additionally, the current of each channel to the smart sensor can be measured. These measurements can then be sent to a control computer. There the power draw can be calculated and, according to different power sources, the voltage drop can be calculated. This voltage drop is then communicated to the power supply which acts accordingly to provide a supply voltage of the modelled power supply. By using this system, various energy sources can be simulated. A block diagram of this prototype is shown in Figure

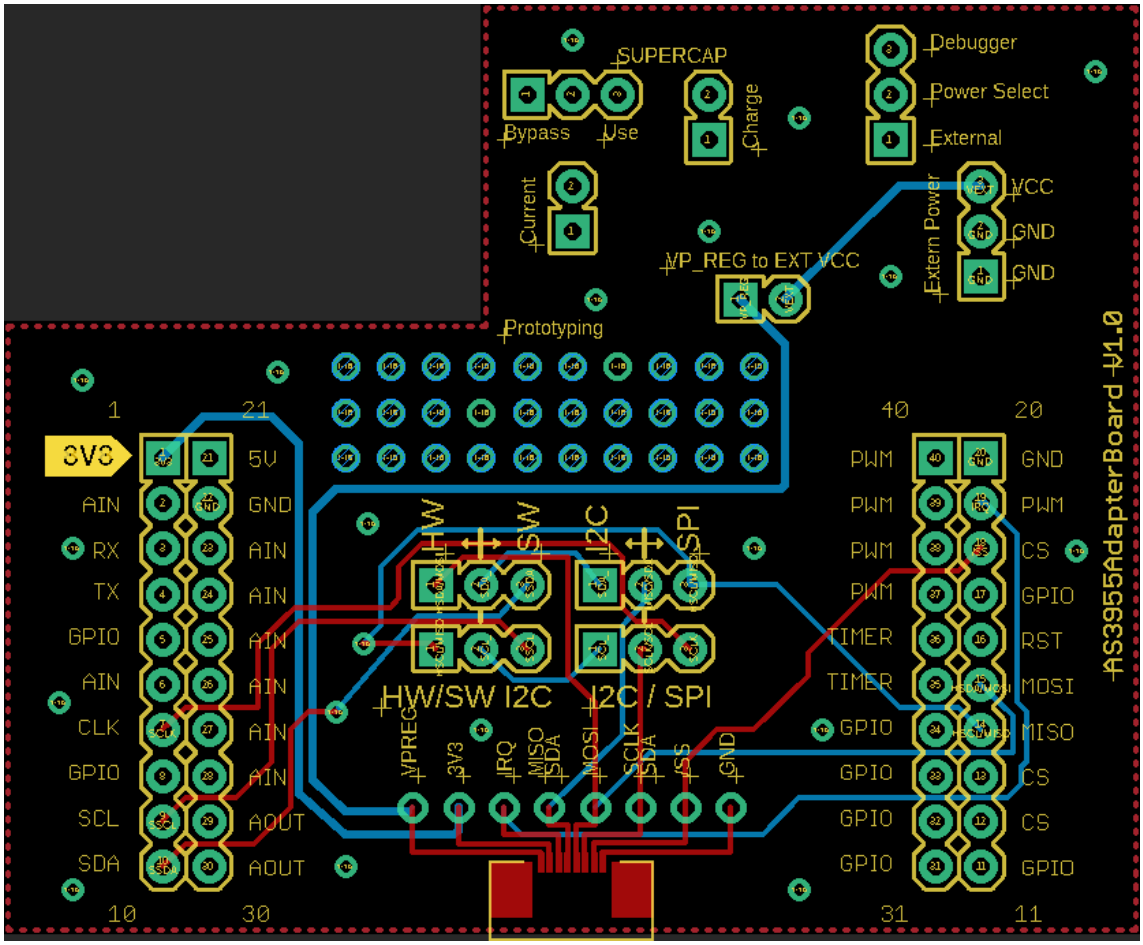

Figure 4.12: Diagram of the expansion PCB of Prototype A.

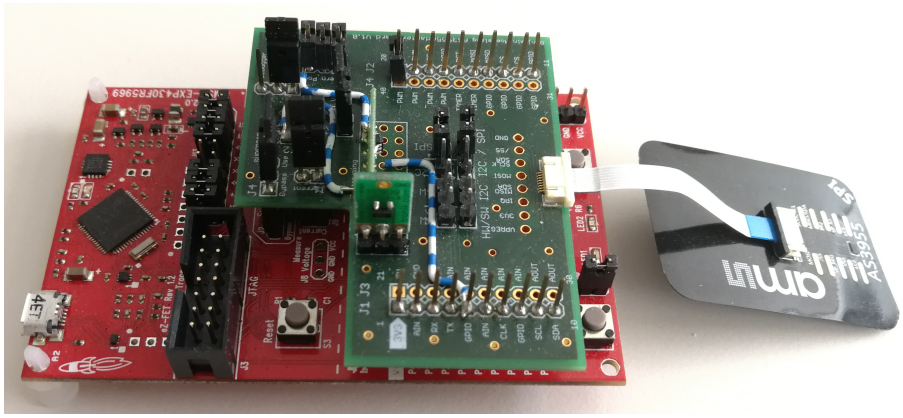

Figure 4.13: Prototype to measure the capabilities of harvesting energy from an NFC field and operating a sensor from a super capacitor.

4.16. The PCB schematic for Prototype B is shown in Figure 4.15. In addition to the two

PCB boards a connector board is added here. This connector board allows the use of the smart sensor without the measurement board.

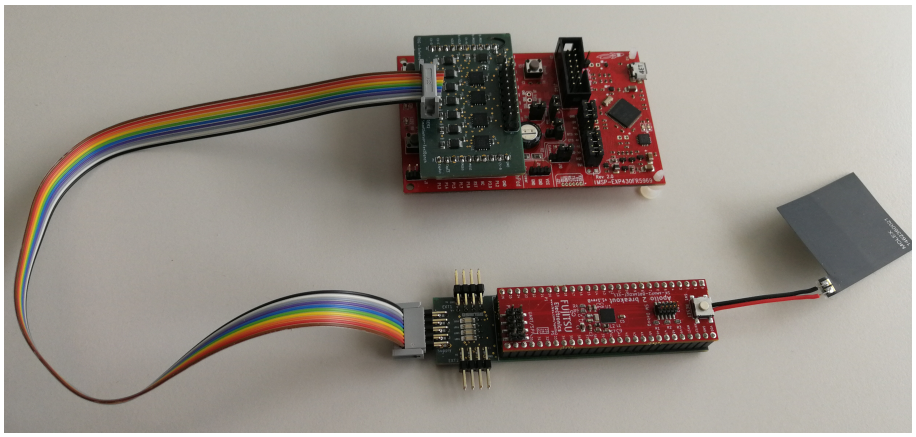

Figure 4.14: Prototype of a secured smart sensor with attached power measurement unit.

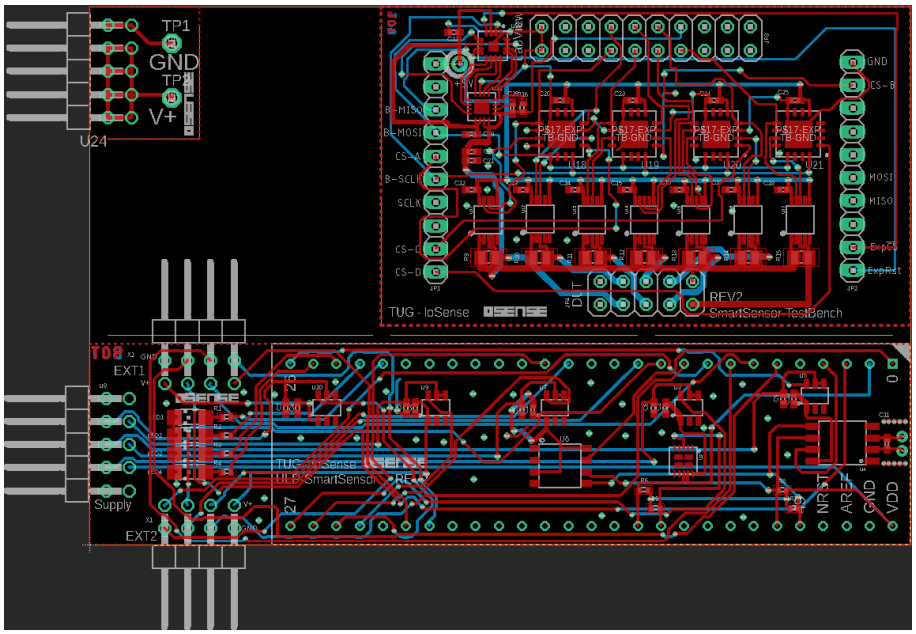

Figure 4.15: Diagram of the PCB design for Prototype B.

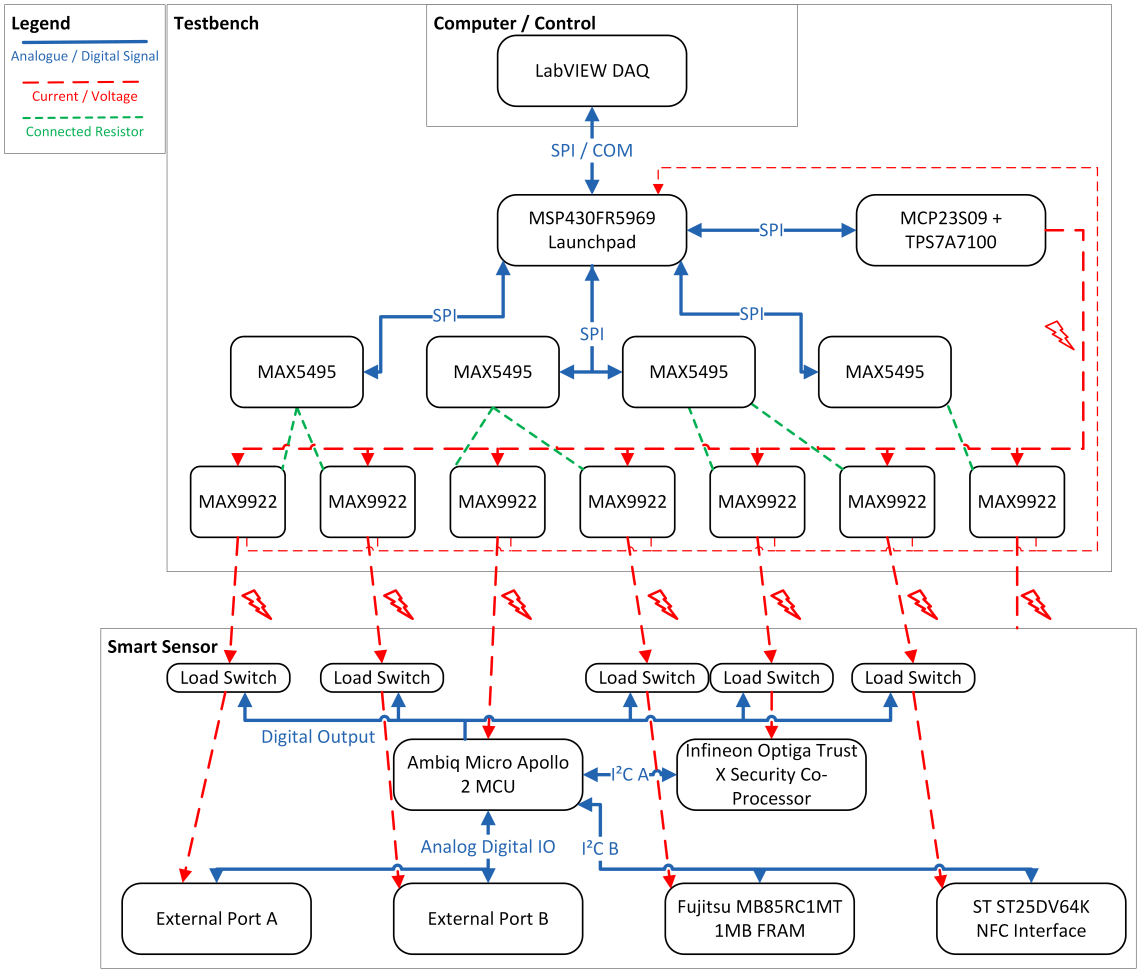

Figure 4.16: Block diagram of Prototype B.

4.4.2 Implementation of the Example Simulation

In the example simulation a robot communicates with a sensor. This sensor is modelled in SystemC. The sensor acquires data from the environment and processes it. When the robot approaches the sensor, an NFC reader is activated. Using the NFC field, the sensor recharges its internal capacitor. The robot then transmits a request to receive the sensor data to the sensor. The sensor processes the request, fetches the data from the internal memory, and sends the data via the NFC interface. When the robot receives the data, it retracts the NFC reader and switches off the NFC field. Figure 4.17 shows a screenshot of the simulation in Gazebo during execution.

As the robot moves towards the sensor while the NFC reader is active, the arm can wobble. This causes differences in the alignment of the NFC antennas and therefore influences the antennas ability to harvest energy. This can be seen in Figure 4.18. This figure shows the explained trace two times. The difference in the two traces lies in the recorded time. This difference is caused by the optimization process described in Section

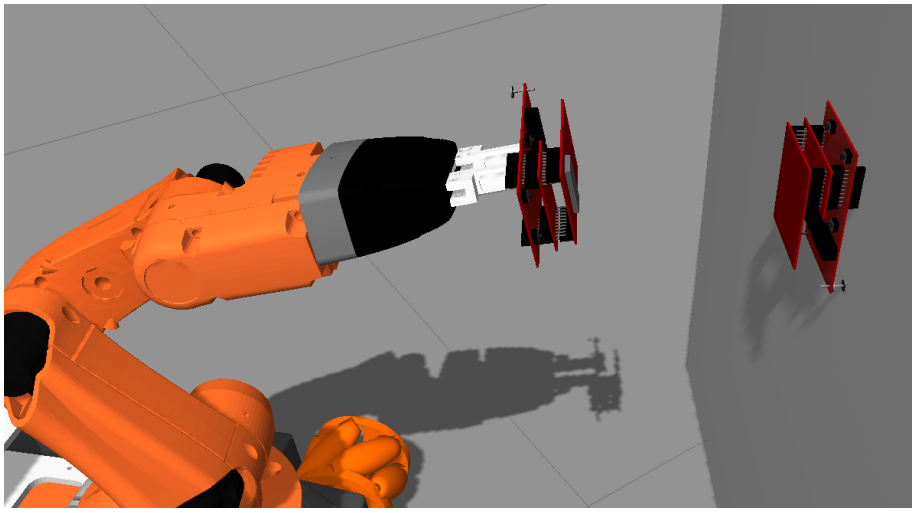

Figure 4.17: Robot holding an NFC reader approaching the sensor.

3.2.3.

In the trace at the top the entire trace is finished in 3.4 ms, while the same trace at the bottom is expanded to 6.5 s. Furthermore, the trace at the top shows an additional data line - *tAdvance*. This data line is used to expand the rest of the data to fit the actual simulation time. The time compression at the top can be seen especially well at the end of the trace. While at the top the decrease in the *Energy* data takes place in 0.1 ms the same decrease uses 50 ms in the bottom trace.

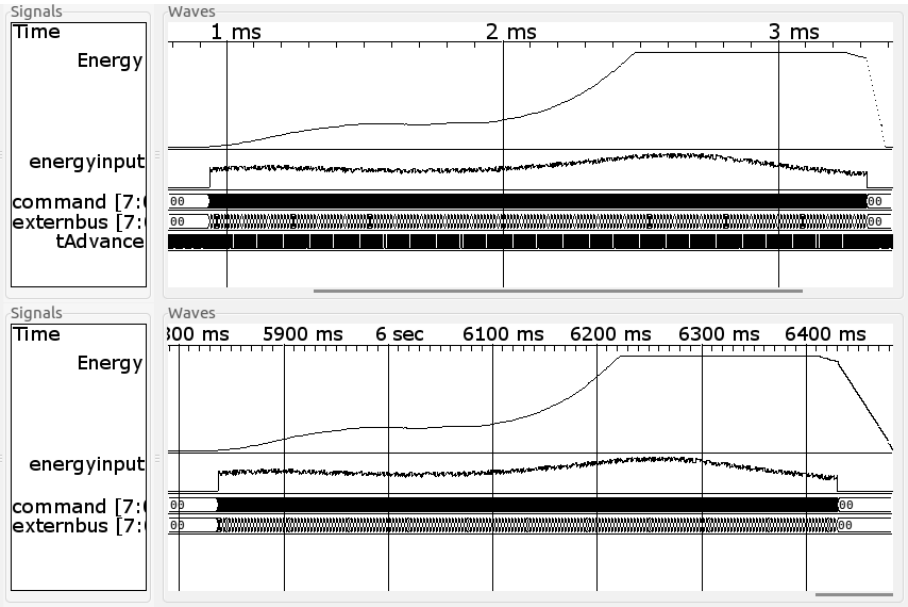

Figure 4.18: Trace from the SystemC simulation. The wobble in the arm position can be seen in the harvested energy trace. The trace is shown two times - top: compressed, bottom: expanded.

4.4.3 Optimization Handling of the Example Simulation

During this simulation, the sensor is idle most of the time. It becomes active when measurements need to be taken or the NFC interface is active. Therefore, the simulation speed can be optimized by prematurely stopping a simulation step and by checking the inputs of the NFC interface. During these idle times, only a counter is active that triggers the measurement procedure. Furthermore, the level of stored energy changes. These two quantities need to be estimated in order to skip simulation time. This estimation is done by using the equations described in Section 3.2. The counter increases linearly over time, whereas the energy level decreases exponentially.

5 Evaluation

To test the developed concept, a proof of concept has been implemented as described in Section 4.4. This chapter focuses on this simulation of this test case and its results. To test the speed improvements, simulation parts are executed with and without the optimizations. The difference between the results shows how effective the optimizations are and how they affect the accuracy of the simulation.

5.1 Results of the Example Simulations

As SystemC does not natively support the skipping of simulation time, a method to support that has been implemented. This section discusses the differences between the raw data and the data that has been post-processed to hide the skipping of time (Section 3.2.3).

This section furthermore discusses the differences introduced by the optimizations of the SystemC simulation speed.

5.1.1 Generated Raw Data and Post-Processed Data

Due to the speed optimizations, the time axis of the raw generated data is warped nonlinearly. Some time steps are executed fully, some are stopped prematurely, and others are not even started. Figure 5.1 shows a comparison between the generated raw data in the top and the post-processed data in the bottom. The arrows in the image connect features in the raw data with the same feature in the post-processed one. It can be seen that the details while the sensor is operating are saved using a better time resolution than while the sensor is idle. This reduces the file-size while keeping the important data.

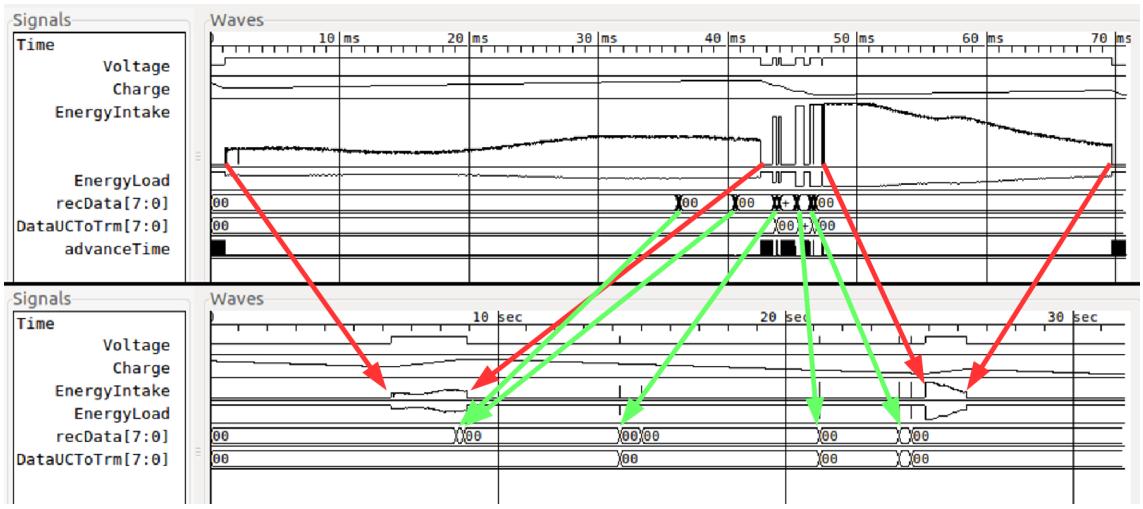

Figure 5.1: The generated data squishes the idle-time and stores the data while operating in more detail.

5.1.2 Differences Between Optimized and Non-optimized Simulations

As the increase of the simulation speed is achieved by prematurely stopping the simulation run and estimating the resulting state, errors in the changing variables are introduced. Thus, the simulation speed increase causes a decreased accuracy.

The simulation speed is given by the real-time-factor (RTF). This is a factor by which the simulation duration has to be multiplied to get the simulated time. To examine the RTF of different sensor operations, simulations that contain only one operation have been performed as micro-benchmarks. The examined sensor operations are:

- **Idle:** The sensor node does not perform any operation. All inputs stay constant.
- **Charge:** The sensor node does not perform any operation. The energy harvesting is active and charges the internal capacitor.
- **Busy:** The sensor node performs calculations or takes a measurement. No external field for the energy harvesting is active.
- **Charge while Busy:** The sensor node is active and may take a measurement. Furthermore, energy harvesting is performed.
- **Field Change:** The sensor node does not perform any operation. The external field for the energy harvesting device changes. The harvested energy changes in each simulation step.
- **Field Change while Busy:** The sensor node is active and may take a measurement. The external field for the energy harvesting and the harvested energy changes in each simulation step.

The simulations for these operations are then executed and the final RTF is calculated. Two levels of optimization have been examined to also show the effects of the optimization.

The *Optimized* version calls the SystemC simulation for every simulation step of Gazebo. Here the simulation is started and stopped when the results can be estimated. The *Heavily Optimized* simulations also reduce the calls to the SystemC simulation by checking the parameter change in the Gazebo simulation. The input parameters are also checked with this optimization and it is evaluated, if the input requires the simulation to be started. Otherwise the results are estimated as described in Section 3.2. Figure 5.2 shows the results of these experiments. The *Unoptimized* simulation always performs the full simulation. This is the baseline of the experiments.

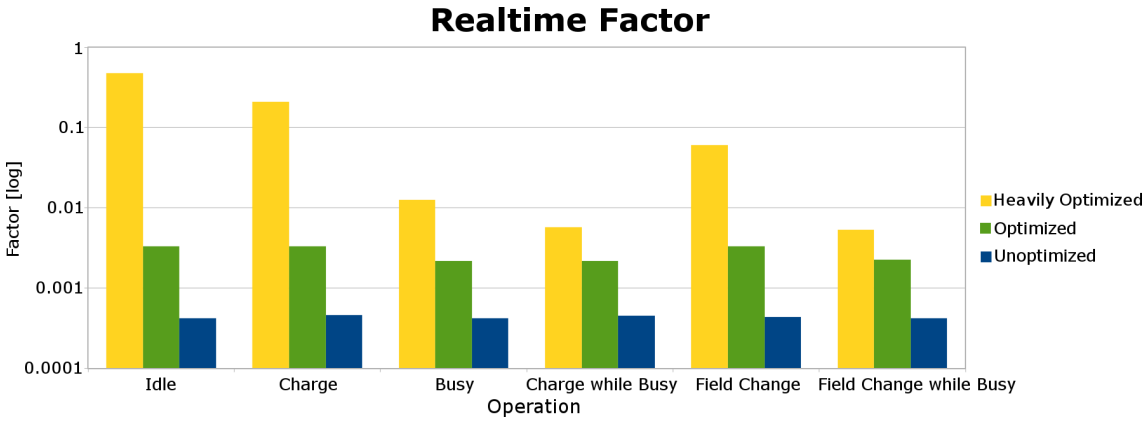

Figure 5.2: The optimization increases the simulation speed. The RTF becomes higher as the simulation becomes faster. Different sensor operations can be optimized by different amounts.

Idle	Charge	Busy	Charge while Busy	Field Change	Field Change while Busy
99,6%	0.1%	0.08%	0.1%	0.06%	0.02%

Table 5.1: Average state occupation during a simulation.

The charge of the sensor's capacitor changes for each simulation step. Therefore, this value needs to be estimated when the simulation skips some steps to optimize the simulation speed. This causes a loss in accuracy when the simulation speed is optimized. Therefore, the difference of the charge of the capacitor directly correlates to the loss in accuracy. As the final error is the sum of the errors throughout the simulation, and each simulation run can simulate a different amount of time, the error is calculated as average error that is made per millisecond. Figure 5.3 shows the average error per millisecond of simulation time for each of the sensor operations. This figure shows that both optimizations cause approximately the same error for all operations. This can be explained by the fact that the optimization uses a linear model to estimate the still occurring changes. This change is not affected by the number of calls to the simulation but only by the simulation time that is estimated. Furthermore, the error increases, when the energy harvesting is active. This signals that some parameters in the model that affect the capacitor charge have not been considered when creating the estimations.

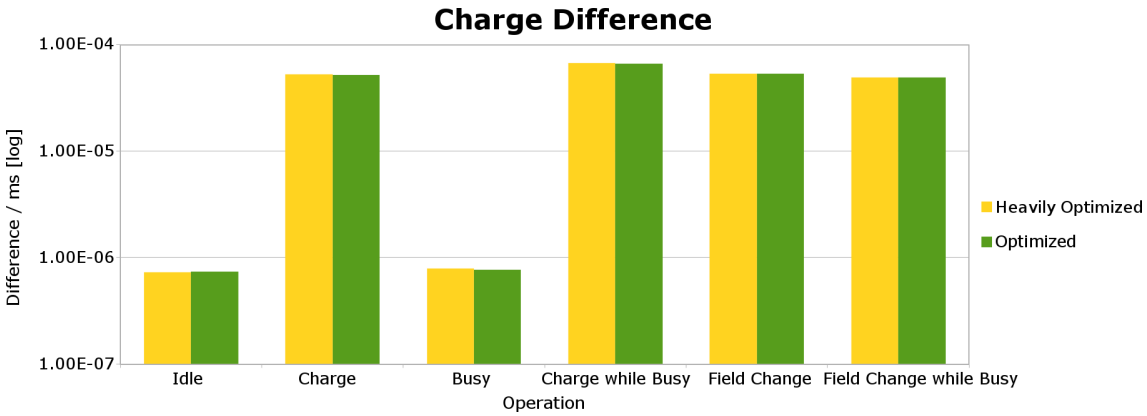

Figure 5.3: The optimization causes a loss in simulation accuracy. This accuracy loss depends on the performed operation and the duration of the operation.

5.2 Validation of Simulation Results

To validate the simulation, its results can be compared to measurements obtained from real hardware. Therefore, prototypes A and B (described in Section 4.4.1) are used to generate data for the evaluation.

As the simulations are generated using data derived from measurements done using the same prototypes, the errors in the simulation only originate from the generalization of the measurement data and the intrinsic errors of the simulator.

5.2.1 Results of the Prototype Measurements and Simulations

Figure 5.4 shows a measurement taken from Prototype A during charging of the capacitor and subsequent operation. A similar scenario has also been simulated. Figure 5.5 illustrates a comparison between the simulation and the real measurement. In this figure the measurement is plotted bold, the simulation using thin lines. The measurement data was aligned to the phases of operation where the change in voltage and current is maximized. This comparison shows that the simulation is close to the real system.

Prototype B can be used to measure the energy consumption of each component of the sensor. Figure 5.6 shows a measurement of the internal currents during an operation. Figure 5.7 also shows the internal currents of the simulated smart sensor during an operation. The comparison between these two figures also shows major similarities. These similarities show that also the simulation of the internal values can be performed using the implemented approach.

Figure 5.6 shows a measurement operation at Prototype B. When the system wakes up, the sensor at *EXT-1* is switched on. During the startup of the external sensor, the system waits. After the startup is finished, the μC switches into another state to acquire the data and stores it in the internal memory. After this the sensor platform is switched back into a low-power mode.

A simulation where a message is received and answered is shown in Figure 5.7. The

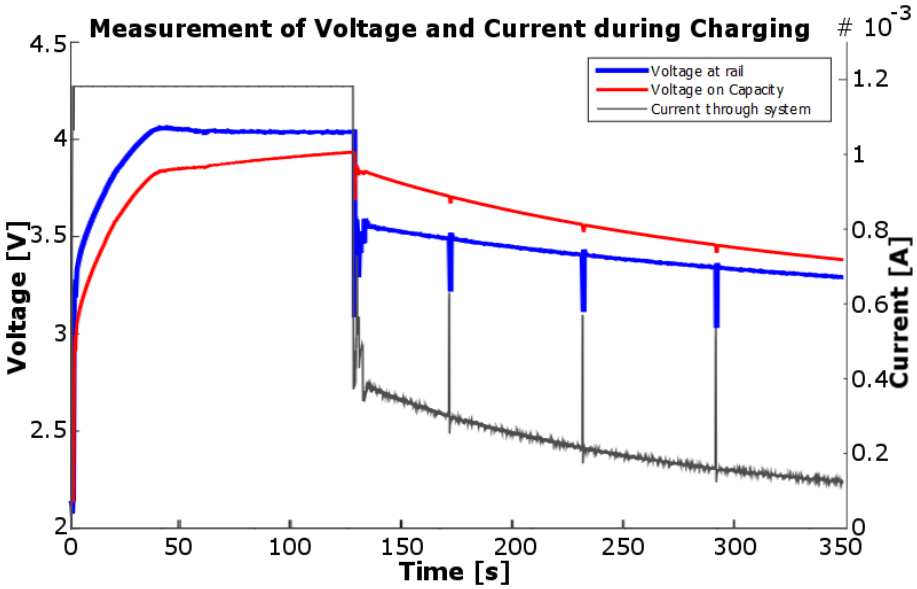

Figure 5.4: Measurement of Prototype A during charging.

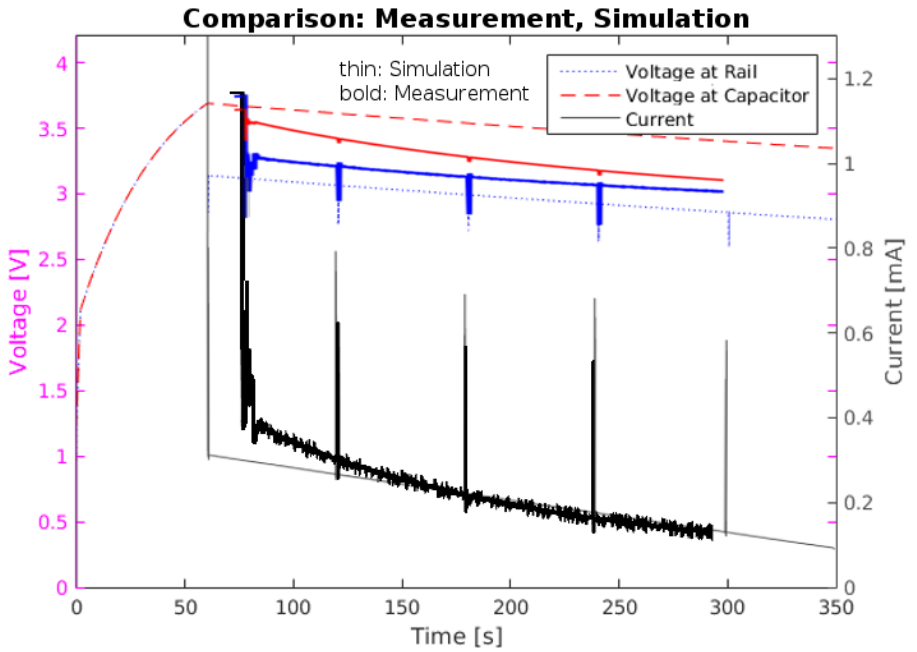

Figure 5.5: Simulation of the charging process of a smart sensor. The measurements of Figure 5.4 are layed over the simulation data to illustrate the error between the simulation and the measurement.

Receiver structure is active at the beginning of the trace. When it recognizes that a valid message has been received, the μC wakes up and receives the data from the *Receiver* structure. When the message is processed, the μC uses the *Memory*. Therefore, the *Memory* uses more energy. When the memory operation is finished, the μC transmits

data to the *Sender* structure. After the data transfer to the *Sender* is finished, the *Sender* switches into a high-power mode to transmit the data.
The trace of the *EnergyLoad* is the sum of all components. The *Voltage* applied to the system changes according to the load applied to the energy provisioning system. As the energy is consumed, the *Charge* drops during these operations.

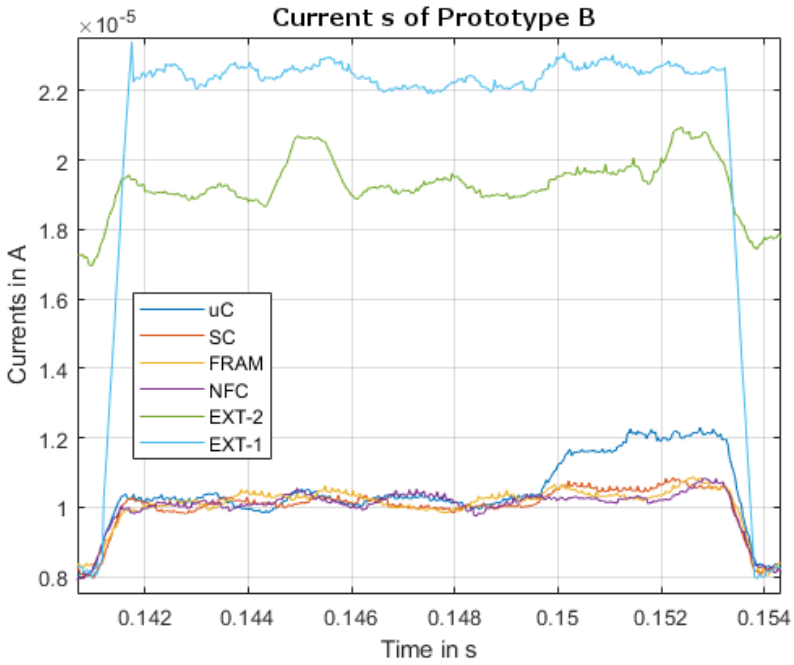

Figure 5.6: Measurements of Prototype B's internal currents during operations.

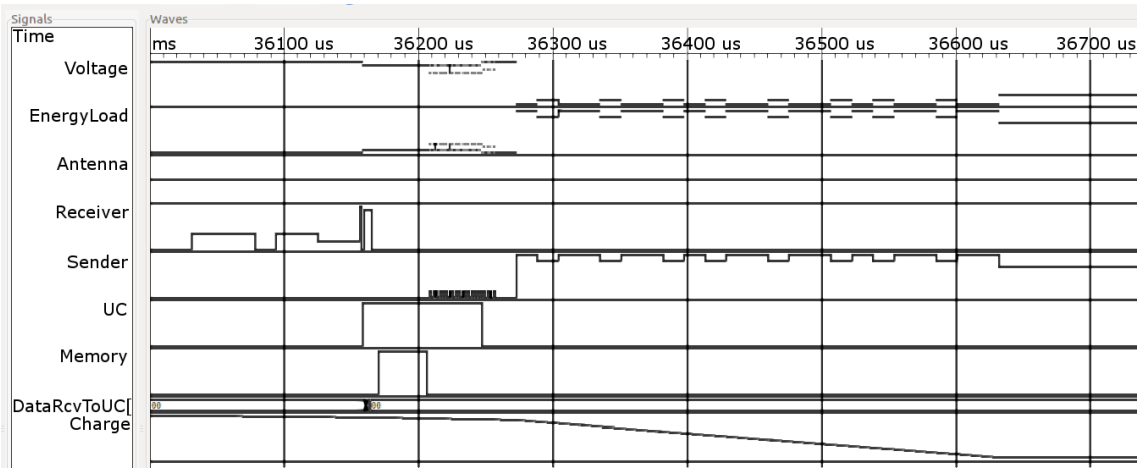

Figure 5.7: Simulation of the internal loads of a smart sensor.

5.3 Discussion of the Results

The results of the previous sections show that the co-simulation concept of connecting an environment simulator and a hardware simulation tool can yield useful information. Such co-simulation has the potential to reduce errors during the creation of the simulation tests as the use case can be used directly to generate stimuli for the new hardware. The shown simulation speed improvements can be used to minimize the simulation duration while the decrease of accuracy of the results is marginal.

The comparison of measurements to a simulation of a similar hardware show that the overall processes of the hardware can still be accurately simulated. Furthermore, the internal behaviour of the simulated hardware can also be stimulated accurately. To achieve more accurate results, the optimization of the simulation can be reduced or switched off.

For a simulation performed during the IoSense project the simulation time reduced from approximately 120 hours to a approximately 12 minutes. Furthermore, the file size to store the simulation traces is also reduced by 91%.

6 Conclusions and Future Work

This chapter intends to draw conclusions from this thesis. Additionally, the intellectual contributions of this thesis are summarized and answers to stated research questions are summarized. Furthermore, the chapter outlines possible future improvements of the simulation concept. This chapter ends by putting forward some recommendations for future research.

6.1 Conclusion

In this thesis a simulation concept has been provided, that can generate stimuli for a hardware simulation out of the use case description. This is done by describing the use case in a robotics simulation tool. This tool can simulate the environment of the hardware to be developed. As this simulation tool is also able to simulate the physical environment around the new hardware, even stimuli for a sensor can be generated. The generated stimuli are then sent to the simulation of the hardware. There the testbench is replaced by the mechanism to receive and distribute the stimuli. Data that is generated in the hardware simulation that can affect the environment of the new hardware is sent back to the environment simulation. There the incoming data is used to perform actions in the environment.
As the simulation of the hardware is slow, improvements to the simulation speed are implemented. The simulation accuracy is reduced to achieve the improved simulation speed. This is done by stopping the simulation of an idle hardware and checking if the simulation needs to be resumed before the simulation step is performed. In order to reduce the introduced errors, the transient variables are estimated.

The shown simulation concept allows the simulation of new hardware in the intended environment. The results indicate that the simulation speed optimizations can achieve a speedup of up to 10^3 for idle simulations and 10^1 for simulations that are continuously busy.

To create the proof of concept simulation and the data needed for the evaluation, two smart sensor prototypes have been realized. These prototypes can be used to characterize the energy consumption of a smart sensor and to characterize an energy provisioning system for the sensor.

6.1.1 New Intellectual Contributions

In this thesis, two research questions have been put forward. These are summarized and answered here.

How to efficiently create stimuli for simulations of Smart Sensors using the use case description?
A new co-simulation concept has been developed that combines the simulation of the hardware and the simulation of the environment. The environment simulation utilizes the use case description to generate stimuli for the hardware simulation. To also stimulate the sensory part of the hardware, a 3D environment including physics simulation has been chosen to simulate the use case.

How to efficiently simulate the complex compound system and environment and smart sensor?
To more efficiently execute the compound simulation, a novel optimization strategy has been proposed and implemented in this thesis. The optimization strategy checks the inputs of the system and decides whether the resulting actions can be estimated or not. If they can be estimated, the simulation is not executed and the state changes are estimated and applied to the system.

6.2 Improvements to the Simulation Concept

There are some possible improvements to the simulator. The communication between the Gazebo simulator and SystemC can be optimized by using binary protocols such as Protobuf. This should reduce the communication overhead.
If the simulation is run on one machine, another improvement can be made by moving the estimation whether the simulation needs to become active to the Gazebo side. As the call of the SystemC simulation needs to perform a context switch for every millisecond of simulated time, this reduction in needed operations has the potential to further increase the simulation speed by improving the estimation process.
A visualization that shows the state of the remote machines can be an improvement for simulations that are performed in a network. This can be an on-line graph of the computed traces. This way an estimation of the state of the simulation is easier.
To validate the hardware in the simulation, a comparison between the input stimuli and the generated traces can be added.
As a further addition the possibility to change the optimization strategy during the simulation can be considered.
To decrease the workload on the engineer creating the simulation, the optimization can learn the input parameters and input changes that require the full simulation to be executed. This can be done by starting the simulation without optimization and checking the results for the simulation steps. If the results can be predicted such inputs allow optimization. If unknown inputs or unknown combinations of inputs are added, the learning process needs to be started again.

6.3 Recommendations for Future Research

The topic of generating stimuli for simulations based on the simulation of the use case is not common in the literature. Thus, further research regarding this topic is advisable. This could further increase the overall simulation speed and reduce the introduced errors. It may also be possible to combine the hardware simulation and the simulation of the environment in a single simulator. This can help the developers of hardware to use the described concept.

As each simulation tool has its individual limitations, further research may also be required to use different HDLs and environmental simulators. Such research could open the developed simulation concept to a wider audience and help in creating better hardware faster.

7 Acknowledgements

This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 692480. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Netherlands, Spain, Austria, Belgium, Slovakia.

IoSense is funded under the agreement number 853326 by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2016 and April 2019. More information https://iktderzukunft.at/en/

I want to thank Infineon Technologies, and especially Holger Bock and Rainer Matischek for providing us the security controllers used in the system creation and for their support that helped to create the prototypes and simulations.

Bibliography

[1] Tristan Gingold. GHDL - An Open Source Compiler and Interpreter for VHDL. https://github.com/ghdl/GHDL, 2017. Last accessed on Jun 06, 2019.

[2] Bleyer. Icarus Verilog for Windows. http://bleyer.org/icarus/, 2000. Last accessed on Jun 06, 2019.

[3] Accelera. SystemC. http://accellera.org/downloads/standards/systemc, 2000. Last accessed on Mar 18, 2019.

[4] Peter S Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[5] Esteban Egea-Lopez, Javier Vales-Alonso, Alejandro S Martinez-Sala, Pablo Pavon-Marino, and Joang García-Haro. Simulation Tools for Wireless Sensor Networks. In *proceedings of the international symposium on performance evaluation of computer and telecommunication systems (SPECTS05)*, page 24, 2005.

[6] S Ying and J Sztipanovits. Foundations for Innovation in Cyber-Physical Systems. In *Workshop Report, Energetics Incorporated, Columbia, Maryland, US*, 2013.

[7] OpenSim Ltd. OMNET++. https://omnetpp.org/. Last accessed on Mar 18, 2019.

[8] Steven McCanne. The LBNL Network Simulator. 1997.

[9] Network Simulator Verion 3. Ns-3. https://www.nsnam.org/, 2019. Last accessed on Nov 04, 2019.

[10] Graphical Network Simulator-3. Gns3. https://gns3.com/, 2018. Last accessed on Nov 04, 2019.

[11] U.S. National Science Foundation. Cyber-Physical Systems (CPS). https://www.nsf.gov/pubs/2019/nsf19553/nsf19553.pdf, 2019. Last accessed on Dec 13, 2019.

[12] IoSense-Consortium. IoSense - Flexible FE/BE Pilot Line for the Internet of Everything. `http://www.iosense.eu`, 2016.

[13] IEEE standard glossary of modeling and simulation terminology, 1989.

[14] ISO/IEC/IEEE International Standard - Systems and software engineering - Developing information for users in an agile environment, 2018.

[15] A. R. Plummer. Model-in-the-Loop Testing. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 220(3):183–199, May 2006.

[16] Udo Brockmeyer, Guido Sandmann, and Michael Beine. Formal Verification Techniques in a Model-Based Development Process based on TargetLink generated C-Code. 01 2006.

[17] Katalin Popovici and Pieter J. Mosterman. *Real-Time Simulation Technologies: Principles, Methodologies, and Applications*. CRC Press, mar 2016.

[18] Jerry Banks, John S Carson, Barry L Nelson, David M Nicol, et al. *Discrete-Event System Simulation*, volume 3. Prentice hall Upper Saddle River, NJ, 1996.

[19] ScienceBySimulation. A chemical reaction modeling and simulation app. https://www.sciencebysimulation.com/chemreax, 2016. Last accessed on Jan 05, 2020.

[20] Patrick Noonan and Robert Berger. Basics of Simulation Technology (SPICE), Virtual Instrumentation and Implications on Circuit and System Design. https://www.ieee.li/pdf/viewgraphs/basics_simulation_technology.pdf, 2007. Last accessed on Jan 05, 2020.

[21] Open Source Robotics Foundation. Gazebo Simulator. http://www.gazebosim.org, 2004. Last accessed on Jan 03, 2017.

[22] Rosen Dinkov. Open Robotic Automation Virtual Environment (OpenRAVE). http://openrave.org/, 2011. Last accessed on Jan 05, 2020.

[23] Rosen Dinkov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, 2010. Last accessed on Jan 05, 2020.

[24] Cyberbotics Ldt. Webots. http://www.cyberbotics.com/, 1996. Last accessed on Jan 05, 2020.

[25] MathWorks. Improving Speed and Accuracy. https://de.mathworks.com/help/physmod/simscape/ug/improving-speed-and-accuracy.html, 2019. Last accessed on Jul 17, 2019.

[26] Gabino Alonso. LTspice: Speed Up Your Simulations. https://www.analog.com/en/technical-articles/ltspice-speed-up-your-simulations.html, n.d. Last accessed on Jan 05, 2020.

[27] Norecopa. Advanced Continuous Simulation Language (ACSL). https://norecopa.no/norina/advanced-continuous-simulation-language-acsl, 2019. Last accessed on Mar 22, 2020.

[28] Borut Zupancic, Rihard Karba, and Drago Matko. *Simulacija dinamicnih sistemov*. Fakulteta za elektrotehniko in racunalnistvo, 1995.

[29] Borut Zupancic. SIMulation of COntinuous Systems (SIMCOS. http://msc.fe.uni-lj.si/Download/Zupancic/Simcos_namestitev.zip. Last accessed on Mar 22, 2020.

[30] Norm Matloff. Introduction to Discrete-Event Simulation and the SimPy Language. *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, 2(2009):1–33, 2008.

[31] David C. Black, Jack Donovan, Bill Bunton, and Anna Keist. A Notion of Time. In *SystemC: From the Ground Up*, pages 59–64. Springer US, December 2009.

[32] David C Black, Jack Donovan, Bill Bunton, and Anna Keist. *SystemC: From the Ground Up*, volume 71. Springer Science & Business Media, 2009.

[33] Nathan Kitchen and Andreas Kuehlmann. Stimulus Generation for Constrained Random Simulation. In *Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '07, page 258–265. IEEE Press, 2007.

[34] Jun Yuan, Kurt Shultz, Carl Pixley, Hillel Miller, and Adnan Aziz. Modeling Design Constraints and Biasing in Simulation Using BDDs. In *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pages 584–590. IEEE Press, 1999.

[35] Wei Wei, Jordan Erenrich, and Bart Selman. Towards Efficient Sampling: Exploiting Random Walk Strategies. In *AAAI*, volume 4, pages 670–676, 2004.

[36] Vibhav Gogate and Rina Dechter. A New Algorithm for Sampling CSP Solutions Uniformly at Random. In *International Conference on Principles and Practice of Constraint Programming*, pages 711–715. Springer, 2006.

[37] Franco Fummi, Giovanni Perbellini, Paolo Gallo, Massimo Poncino, Stefano Martini, and Fabio Ricciato. A Timing-Accurate Modeling and Simulation Environment for Networked Embedded Systems. In *Proceedings of the 40th annual Design Automation Conference*, pages 42–47. ACM, 2003.

[38] Florian Schloegl, Sebastian Rohjans, Sebastian Lehnhoff, Jorge Velasquez, Cornelius Steinbrink, and Peter Palensky. Towards a Classification Scheme for Co-Simulation Approaches in Energy Systems. In *2015 International Symposium on Smart Electric Distribution Systems and Technologies (EDST)*, pages 516–521. IEEE, 2015.

[39] Cornelius Steinbrink, Sebastian Lehnhoff, Sebastian Rohjans, Thomas I Strasser, Edmund Widl, Cyndi Moyo, Georg Lauss, Felix Lehfuss, Mario Faschang, Peter Palensky, et al. Simulation-based Validation of Smart Grids - Status Quo and Future Research Trends. In *International Conference on Industrial Applications of Holonic and Multi-Agent Systems*, pages 171–185. Springer, 2017.

[40] Torsten Blochwitz, Martin Otter, Martin Arnold, Constanze Bausch, H Elmqvist, A Junghanns, J Mauß, M Monteiro, T Neidhold, Dietmar Neumerkel, et al. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*, number 063, pages 105–114. Linköping University Electronic Press, 2011.

[41] Jérôme Hugues, Jean-Marie Gauthier, and Raphaël Faudou. Integrating AADL and FMI to Extend Virtual Integration Capability. *arXiv preprint arXiv:1802.05620*, 2018.

[42] Stefano Centomo, Julien Deantoni, and Robert De Simone. Using SystemC Cyber Models in an FMI Co-Simulation Environment. 2016.

[43] Silvio Traversaro, Luca Tricerri, and Prashanth Damadoss. Gazebo-FMI. https://github.com/robotology/gazebo-fmi, 2018. Last accessed on Jan 25, 2020.

[44] Zhenkai Zhang, Joseph Porter, Emeka Eyisi, Gabor Karsai, Xenofon Koutsoukos, and Janos Sztipanovits. Co-simulation framework for design of time-triggered cyber physical systems. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems - ICCPS '13*. ACM Press, 2013.

[45] Rolf Isermann, Jochen Schaffnit, and Stefan Sinsel. Hardware-in-the-loop simulation for the design and testing of engine-control systems. *Control Engineering Practice*, 7(5):643–653, 1999.

[46] Darcy Bullock, Brian Johnson, Richard B Wells, Michael Kyte, and Zhen Li. Hardware-in-the-loop simulation. *Transportation Research Part C: Emerging Technologies*, 12(1):73–89, 2004.

[47] M. Bacic. On hardware-in-the-loop simulation. In *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, 2005.

[48] Stefan Kowalewski, Bernhard Rumpe, and Andre Stollenwerk. Cyber-Physical Systems - eine Herausforderung an die Automatisierungstechnik? *arXiv preprint arXiv:1409.0385*, 2014.

[49] Thomas Bock, Markus Maurer, Franciscus Meel, and Thomas Müller. Vehicle in the Loop. *ATZ - Automobiltechnische Zeitschrift*, 110(1):10–16, January 2008.

[50] Guy Berg and Berthold Färber. Vehicle in the Loop. In *Handbuch Fahrerassistenzsysteme*, pages 155–163. Springer Fachmedien Wiesbaden, 2015.

[51] H. El Tahawy, D. Rodriguez, S. Garcia-Sabiro, and J.-J. Mayol. VHD/sub e/LDO: A new mixed mode simulation. In *Proceedings of EURO-DAC 93 and EURO-VHDL 93- European Design Automation Conference*. IEEE Comput. Soc. Press.

[52] Youngmin Yi, Dohyung Kim, and Soonhoi Ha. Fast and Accurate Cosimulation of MPSoC Using Trace-Driven Virtual Synchronization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(12):2186–2200, 2007.

[53] Raphael Beese. Prototypische Umsetzung eines XiL-Testsystems für Gesamtfahrzeugintegration und -absicherung. https://www.fb06.fh-muenchen.de/fb/images/img_upld/arbeiten/00011.pdf, 2007.

[54] Meng-Huan Wu, Cheng-Yang Fu, Peng-Chih Wang, and Ren-Song Tsay. An Effective Synchronization Approach for Fast and Accurate Multi-core Instruction-set Simulation. In *Proceedings of the seventh ACM international conference on Embedded software*, pages 197–204, 2009.

[55] Shafagh Jafer, Qi Liu, and Gabriel Wainer. Synchronization methods in parallel and distributed discrete-event simulation. *Simulation Modelling Practice and Theory*, 30:54–73, January 2013.

[56] Sunghee Lee, Bueng Il Hwang, Kang-Bok Seo, and Woo Jin Lee. Relative Time Synchronization of Distributed Applications for Software-in-the-Loop Simulation. In *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*. IEEE, August 2016.

[57] Yehuda Naveh, Michal Rimon, Itai Jaeger, Yoav Katz, Michael Vinov, Eitan s Marcu, and Gil Shurek. Constraint-based Random Stimuli Generation for Hardware Verification. *AI magazine*, 28(3):13–13, 2007.

[58] Robert Wille, Daniel Große, Finn Haedicke, and Rolf Drechsler. SMT-based Stimuli Generation in the SystemC Verification Library. In *2009 Forum on Specification & Design Languages (FDL)*, pages 1–6. IEEE, 2009.

[59] Shuo Yang, Robert Wille, Daniel Grosse, and Rolf Drechsler. Minimal Stimuli Generation in Simulation-Based Verification. In *2013 Euromicro Conference on Digital System Design*. IEEE, September 2013.

[60] Robert Oshana and Mark Kraeling. *Software Engineering for Embedded Systems: Methods, Practical Techniques, and Applications*. Newnes, 2019.

[61] Tingting Yu. Testing Embedded Systems Applications, 2010.

[62] F Bouchhima, M Briere, G Nicolescu, M Abid, and EM Aboulhamid. A SystemC/Simulink co-simulation framework for continuous/discrete-events simulation. In *2006 IEEE International Behavioral Modeling and Simulation Workshop*, pages 1–6. IEEE, 2006.

[63] Francisco Mendoza, Christian Köllner, Jürgen Becker, and Klaus D Müller-Glaser. An Automated Approach to SystemC/Simulink Co-Simulation. In *2011 22nd IEEE International Symposium on Rapid System Prototyping*, pages 135–141. IEEE, 2011.

[64] Massimo Bombana and Francesco Bruschi. SystemC-VHDL co-simulation and synthesis in the HW domain. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 101–105. IEEE, 2003.

[65] Davide Quaglia, Riccardo Muradore, Roberto Bragantini, and Paolo Fiorini. A SystemC/Matlab co-simulation tool for networked control systems. *Simulation Modelling Practice and Theory*, 23:71–86, 2012.

[66] Christoph Roth, Harald Bucher, Simon Reder, Florian Buciuman, Oliver Sander, and Juergen Becker. A SystemC Modeling and Simulation Methodology for Fast and Accurate Parallel MPSoC Simulation. In *2013 26th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6. IEEE, 2013.

[67] Seunghyun Park, Hanjoo Kim, Hichan Moon, Jun Heo, and Sungroh Yoon. Concurrent Simulation Platform for Energy-Aware Smart Metering Systems. *IEEE transactions on Consumer Electronics*, 56(3):1918–1926, 2010.

[68] Aline Mello, Isaac Maia, Alain Greiner, and Francois Pecheux. Parallel Simulation of SystemC TLM 2.0 Compliant MPSoC on SMP Workstations. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 606–609. European Design and Automation Association, 2010.

[69] Christoph Schumacher, Rainer Leupers, Dietmar Petras, and Andreas Hoffmann. parSC: Synchronous Parallel SystemC Simulation on Multi-Core Host Architectures. In *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 241–246. IEEE, 2010.

[70] Rohit Sinha, Aayush Prakash, and Hiren D Patel. Parallel Simulation of Mixed-abstraction SystemC Models on GPUs and Multicore CPUs. In *17th Asia and South Pacific Design Automation Conference*, pages 455–460. IEEE, 2012.

[71] Jan Henrik Weinstock, Christoph Schumacher, Rainer Leupers, Gerd Ascheid, and Laura Tosoratto. Time-Decoupled Parallel SystemC Simulation. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–4. IEEE, 2014.

[72] Asad Khan and Chris Wolf. Simics/SystemC Hybrid Virtual Platform - A Case Study. http://www2.dac.com/50th/proceedings/slides/13D_4.pptx, 2013. Last accessed on Jul 22, 2019.

[73] Asad Khan, Weiqiang Ma, Chris Wolf, and Bengt Werner. Multi-Threaded Simics SystemC Virtual Platform. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 373–379. IEEE Press, 2015.

[74] Peter Garraghan, David McKee, Xue Ouyang, David Webster, and Jie Xu. SEED: A Scalable Approach for Cyber-Physical System Simulation. *IEEE Transactions on Services Computing*, 9(2):199–212, 2015.

[75] SJ Clement, David Wesley McKee, Richard Romano, Jie Xu, JM Lopez, and David Battersby. The Internet of Simulation: Enabling Agile Model Based Systems Engineering for Cyber-Physical Systems. In *2017 12th System of Systems Engineering Conference (SoSE)*, pages 1–6. IEEE, 2017.

[76] Samy Meftali, JOEL Vennin, and Jean-Luc Dekeyser. A fast SystemC simulation methodology for Multi-Level IP/SoC design. In *IFIP Intl. Workshop on IP Based SoC Design*, 2003.

[77] Gunar Schirner, Andreas Gerstlauer, and Rainer Dömer. Multifaceted Modeling of Embedded Processors for System Level Design, Abstract. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, 2007.

[78] Christoph Ossimitz. The HW/SW Design Language SystemC. https://ti.tuwien.ac.at/ecs/teaching/courses/hwswcode_vu/hwsw-codesign-student-presentations/4-systemc.pdf, 2016. Last accessed on Jul 17, 2019.

[79] Syed Saif Abrar, Maksim Jenihhin, and Jaan Raik. SystemC-Based Loose Models for Simulation Speed-Up by Abstraction of RTL IP Cores. In *2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, pages 71–74. IEEE, 2015.

[80] Matthias Jung, Christian Weis, and Norbert Wehn. DRAMSys: A Flexible DRAM Subsystem Design Space Exploration Framework. *IPSJ Transactions on System LSI Design Methodology*, 8:63–74, 2015.

[81] Feihong Xia, Philip Griefnow, Serge Klein, Raul Tharmakulasingam, Andreas Balazs, Matthias Thewes, and Jakob Andert. Crank Angle Resolved Real-Time Engine Modeling for HiL Based Component Testing. 10 2017.

[82] Henning Zabel, Wolfgang Müller, and Andreas Gerstlauer. Accurate RTOS Modeling and Analysis with SystemC. In *Hardware-dependent Software*, pages 233–260. Springer, 2009.

[83] Shai Fine, Ari Freund, Itai Jaeger, Yishay Mansour, Yehuda Naveh, and Avi Ziv. Harnessing machine learning to improve the success rate of stimuli generation. *IEEE Transactions on Computers*, 55(11):1344–1355, 2006.

[84] Olga Goloubeva, M Sonza Reorda, and Massimo Violante. Automatic Generation of Validation Stimuli for Application-Specific Processors. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 1, pages 188–193. IEEE, 2004.

[85] Finn Haedicke, Hoang M Le, Daniel Große, and Rolf Drechsler. CRAVE: An Advanced Constrained RAndom Verification Environment for SystemC. In *2012 International Symposium on System on Chip (SoC)*, pages 1–7. IEEE, 2012.

[86] Kazuaki Maeda. Performance Evaluation of Object Serialization Libraries in XML, JSON and Binary Formats. In *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*. IEEE, May 2012.

[87] Thomas W. Pieber, Thomas Ulz, Christian Steger, and Rainer Matischek. Hardware Secured, Password-based Authentication for Smart Sensors for the Industrial Internet of Things. In *Network and System Security*, pages 632–642. Springer International Publishing, 2017.

[88] Thomas W. Pieber, Thomas Ulz, and Christian Steger. SystemC Test Case Generation with the Gazebo Simulator. In *Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pages 67–72. SCITEPRESS - Science and Technology Publications, 2017.

[89] Thomas W. Pieber, Thomas Ulz, and Christian Steger. Using Gazebo to Generate Use Case Based Stimuli for SystemC. In *Advances in Intelligent Systems and Computing*, pages 241–256. Springer International Publishing, November 2018.

[90] Thomas Wolfgang Pieber, Thomas Ulz, and Christian Steger. Model-Based Design of Secured Power Aware Smart Sensors. In *Sensor Systems Simulations*, pages 227–251. Springer International Publishing, June 2019.

[91] Thomas W. Pieber, Benjamin Mößlang, Thomas Ulz, and Christian Steger. Towards Continuous Sensor Operation: Modelling a Secured Smart Sensor in a Sparse Network Operated by Energy Harvesting. In *Proceedings of the 9th International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2019)*, pages 57–64. SCITEPRESS - Science and Technology Publications, 2019.

[92] Thomas Wolfgang Pieber, Fikret Basic, Thomas Ulz, and Christian Steger. Simulating a Network: An Approach for Connecting Multiple SystemC Simulations. In *SENSORCOMM 2019, The Thirteenth International Conference on Sensor Technologies and Applications*, pages 21–26. IARIA, October 2019.

A
Publications

The following publications have been created during the conduction of this thesis. They are collected in this chapter.

P1 Thomas W. Pieber, Thomas Ulz, Christian Steger and Rainer Matischek. Hardware Secured, Password-based Authentication for Smart Sensors for the Industrial Internet of Things. In Network and System Security, pages 632-642. Springer International Publishing, 2017. [87]

P2 Thomas W. Pieber, Thomas Ulz, and Christian Steger. SystemC Test Case Generation with the Gazebo Simulator. In Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications, pages 67-72. SCITEPRESS - Science and Technology Publications, 2017. [88]

P3 Thomas W. Pieber, Thomas Ulz and Christian Steger. Using Gazebo to Generate Use Case Based Stimuli for SystemC. In Advances in Intelligent Systems and Computing, pages 241-256. Springer International Publishing, November 2018. [89]

P4 Thomas W. Pieber, Thomas Ulz and Christian Steger. Model-based Design of Secured Power Aware Smart Sensors. In Sensor System Simulations, pages 241-256. Springer International Publishing, June 2019. [90]

P5 Thomas W. Pieber, Benjamin Moesslang, Thomas Ulz and Christian Steger. Towards Continuous Sensor Operation: Modelling a Secured Sensor in a Sparse Network Operated by Energy Harvesting. In Proceedings of the 9th International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2019), pages 57-64, SCITEPRESS - Science and Technology Publications, 2019. [91]

P6 Thomas W. Pieber, Fikret Basic, Thomas Ulz and Christian Steger. Simulating a Network: An Approach for Connecting Multiple SystemC Simulations. In SENSORCOMM 2019, The Thirteenth International Conference on Sensor Technologies and Applications, pages 21-26, IARIA, October 2019. [92]

Hardware Secured, Password-based Authentication for Smart Sensors for the Industrial Internet of Things

Thomas W. Pieber[1], Thomas Ulz[1], Christian Steger[1], and Rainer Matischek[2]

[1] Graz University of Technology - Institute for Technical Informatics, Graz, Austria
{thomas.pieber, thomas.ulz ,steger}@tugraz.at
[2] Infineon Technologies Austria AG, Graz, Austria
rainer.matischek@infineon.com

Abstract. Sensors are a vital component for the *Internet of Things*. These sensors gather information about their environment and pass this information to control algorithms and/or actuators. To operate as effective as possible the sensors need to be reconfigurable, which allows the operators to optimize the sensing activities. In this work we focus on the mechanisms of such reconfiguration possibilities. As the reconfiguration can also be used to manipulate the sensors (and their attached systems) in a subtle way, the security of the reconfiguration interface is of utmost importance. Within this work we test a lightweight authentication method for use on a smart sensor and describe a possible implementations of the authentication mechanism on a hardware security module.

1 Introduction

In the Internet of Things (IoT) sensors are key components. They create the bulk of the information needed to control their environment according to the wishes of the operator. They furthermore monitor the environment and need to make decisions if the monitored environment is changing in a way that needs the operators' attention. The sensors are equipped with some sort of microcontroller, software, energy source, communication mechanism, and most likely an interface for configuring the software and the decision making. This interface poses a threat to the integrity and trustworthiness of the sensor itself and, in the long run, to the whole system. In order to become a trustworthy system the configuration- and sensor data must be protected against all adversaries. To accomplish this, the sensors can be equipped with tamper resistant hardware security modules (hereafter HSM or security controller). This HSM on a smart sensor can perform critical operations during the configuration of the device and during communication of sensor data to the outside world. These critical operations include the encryption of sensor data, establishing a secured and authenticated channel to maintenance personnel, and the secured storage of configuration- and authentication data and cryptographic keys.

Original work published in Network and System Security, pages 632-642. Springer International Publishing, 2017.

The authentication of users that can see and manipulate the confidential settings of the sensor is one of the core features that a secured system needs. This process not only blocks others from accessing the confidential information but also enforces that only trusted personnel can manipulate the settings. One of the most convenient and widely used methods of authentication is the use of passwords. These passwords can be remembered by the trusted operators and take the function of a shared secret. In a conventional system the user shows the knowledge of the shared secret by directly entering it on the used device. In the case of remote authentication, as it is the case with smart sensors, the password must be transmitted securely to the verifying party. In an unconstrained device this would be accomplished by securing the channel against eavesdroppers and then sending the password over the secured channel. For resource constrained devices this method is impractical. Therefore, methods that perform the authentication alongside the establishment of the secured channel have been developed. This is not only faster than performing these operations sequentially, but also more efficient in terms of energy usage, computation steps needed, and memory allocated on the constrained device. At this step an HSM with dedicated cryptographic hardware can also perform these steps faster, more efficient, and in a more secured fashion compared to a normal microcontroller. The HSM can additionally store the users' credentials in the tamper resistant memory, keeping information leakage as low as possible.

A HSM is typically very limited in terms of general computational power and available memory. This entails that the used protocols must be lightweight in those parameters. This is additionally challenged by the energy constraints on the smart sensor. As such sensors might be operated using battery power, the cryptographic challenging (and therefore energy hungry) functions must be reduced to a minimum to keep the sensor alive as long as possible.

In this work we examined the use of a lightweight authentication method for smart sensors. Therefore, we used simulation techniques to design and implement a prototype application and tested the results on an Infineon type security controller.

The remainder of this paper is structured as follows: In Section 2 the prerequisites for the implementation and related works are stated. The design questions and the approaches are elaborated in Section 3. Details on the implementation are given in Section 4. The 5^{th} section is dedicated to the evaluation of the authentication protocol and answers why the used protocol and hardware is suitable for smart sensors. Possible future work is stated in Section 6. The paper concludes with Section 7.

2 Related Work

To perform a secured authentication a key agreement protocol needs to be used. the most widely used protocol for this task is the Diffie-Hellman key exchange [1]. This protocol defines public and private keys. After the exchange of the public keys a shared secret can be calculated. The security of that scheme is based on

the Computational Diffie-Hellman problem. However, this protocol alone is not able to authenticate the communicating parties.

The authentication of users is a crucial part of a secured communication as an adversary can impersonate the other communication partner and perform a man-in-the-middle attack. Typically, the authentication is done with a shared secret - a password. There are many algorithms that use passwords for authentication. One of them is proposed by M. Peyravian and N. Zunic [2]. This protocol is especially well-suited for use in microcontrollers as the only cryptographic function is a collision-resistant hash function H and therefore uses very little computational effort. In this protocol the user sends his username (un) and a nonce (ru) to the server which replies with another nonce (rs). The user, knowing the password (pw), calculates the function M = H(H(un, pw), ru, rs) and sends the resulting M to the server. The server then uses a lookup-table to get the H(un, pw) matching to the username and can verify M. This protocol can authenticate a user against the server but does not provide the needed security on its own. This has to be done beforehand with a key exchange. This, on the other hand, needs more computational time as the two protocols need to be executed. Another, newer protocol was proposed by I. Liao, C. Lee, and M. Hwang [3]. In this protocol a message pair is exchanged during the registration and further three messages need to be sent for the authentication. This protocol, when executed with ECC, requires four multiplications, two additions, five hash-operations and one random number. Additionally to those operations, the operations for securing the channel in the first place have to be added.

There are also protocols that perform a key agreement while authenticating the user to the server. One of the first protocols that perform an authenticated key exchange is the EKE protocol proposed by S. M. Bellowin and M. Merrit proposed [4]. This protocol is the predecessor to most modern protocols. It works by symmetrically encrypting a public key and session key with the password. This is also the weak part of this protocol as S. Halevi and H. Krawczyk [5]. They say that it is not wise to use a password (or any other low-entropy key) as a key to a cryptographic function. The term “low-entropy” means that even a random string of ASCII-characters uses only the letters from 20 to 126 (=106) of all 256 possible 8-bit characters. This means that more than half of the possibilities are not used and therefore such strings should not be used for encryption, only for authentication.

There are many variants of this protocol in the literature such as [6–10]. The Gennaro-Lindell PAKE protocol from [10] can be proven secure in the standard model of cryptography. A computationally less expensive protocol is the SPAKE-protocol from [6]. This protocol was proposed by M. Abdalla and D. Pointcheval and is proven to be secure under the random oracle model. It is also very efficient as it only needs two messages to be exchanged. The whole protocol, when implemented with ECC, uses only six multiplications, two additions, five hash-operations, and one random number. Abdalla states in [11] that:

> [...] the simple password-authenticated key exchange protocol [...] to which we refer as SPAKE [...] is among the most efficient PAKE schemes based on the EKE protocol.

3 Design

There are two important questions to be answered in order to design an authentication mechanism for smart sensors. (I) In which situation of the sensor's lifecycle is this authentication going to be used? (II) What hardware can be used in order to perform the critical steps - and what costs / benefits come with that hardware?

The answers to these questions are intertwined. The configuration of the sensor system should be able to be performed at any time during the sensor's lifecycle. That means that the configuration interface might not be connected to any power source or even the controller storing the configuration data is not connected at all (e.g. during the production of the system parts). This leads to the use of Near Field Communication (NFC) as a communication mechanism and energy source for the chip containing the (confidential) parameters. This entails that a security controller that is capable of using an NFC antenna is needed. One controller that is capable of such operation is used in [12]. The use of this controller furthermore comes with the benefit that the cryptographic functions, necessary during the communication, can be performed with the harvested energy and do not consume the limited energy source on board of the sensor.

Another requirement that comes from using the same authentication method throughout the entire lifecycle of the sensor, is the possibility of changing cryptographic keys, encryption parameters, usernames and passwords. This can be subsumed with the term *Bring Your Own Key / Encryption (BYOK / BYOE)* [13]. With the use of such methods it can be assured that the device cannot be read by anyone except the authorized persons. This rises the need for a secure user authentication scheme. The authentication of the sensor itself can be performed by only sharing the password with one only one sensor.

The SPAKE-protocol introduced in [6] is a suitable protocol for establishing an authenticated secure link between the trustworthy sensor and the user. This protocol performs a Diffie-Hellman key agreement to generate the session key for the encrypted messages. To authenticate the user a mask is calculated that is applied to the public keys of both partners. At the remote end the mask is removed and the key agreement finishes. If the calculated masks do not match the resulting key will be different and the communication cannot be performed.

Because of the reduced memory consumption on the smart sensor and the sufficient hardware acceleration of the secure element, elliptic curve cryptography (ECC) was chosen to perform the key agreement and authentication. In Table 1 the SPAKE2 algorithm using ECC is shown. There $K_A \stackrel{!}{=} K_B$ and therefore secret symmetric the key $sk_A = sk_B$.

Table 1. Design of the ECC implementation of the SPAKE2 algorithm [6]

public information: $G, H(\cdot), u_A, u_B$
private shared information: $password$

User A	User B
$x = rand()$	$y = rand()$
$X = xG;\ M = H(u_A)G$	$Y = yG;\ N = H(u_B)G$
$X^* = X + (password)M$	$Y^* = Y + (password)N$
$X^* \rightarrow$	
$\leftarrow Y^*$	
$N = H(u_B)G$	$M = H(u_A)G$
$K_A = x(Y^* + Inv((password)N))$	$K_B = y(X^* + Inv((password)M))$
$sk_A = H(H(u_A), H(u_B), X^*, Y^*, password, K_A)$	
	$sk_B = H(H(u_A), H(u_B), X^*, Y^*, password, K_B)$

The use of a special hardware for the authentication yields some constraints on what operations can be used to perform the authentication. Such constraints can come from be the available memory of the HSM, the computational speed, the energy intake during the computation, and the hardware support of cryptographic functionalities.

The selected HSM supports ECC operations up to 256 bits and SHA256. Therefore, the final implementation only supports curves with 256-bit parameters and uses the SHA256 algorithm to perform the final key generation and key expansion for the username and password. The constraints on the memory entail a maximum of different curves and users.

3.1 Evaluation Design

The evaluation of the implementation against the reference from Googles Weave project [14] cannot be performed, as currently fundamental differences between that implementation and the description of the protocol in [6] exist. Most notably Googles design uses fixed points for M and N and they do not use a Key-Derivation-Function to generate the session key - this is a mistake denoted in [6] that would render the authentication insecure. Furthermore, Google uses 224-bit parameters while this implementation of SPAKE2 uses 256-bit. Therefore, the evaluation will concentrate on the performance of our implementation of the SPAKE2-protocol on the HSM.

To evaluate the performance of the protocol the security controller communicates with an implementation on a laptop computer over an NFC interface. To get a complete picture of the protocol-performance the round-trip time for packets with different lengths is the baseline. For further testing a version of the SPAKE2 protocol was deployed on an NFC enabled Android device. There the whole process for authentication and writing sample configurations was monitored.

Fig. 1. Prototype-hardware used for evaluating the performance

For further comparison, two variants of the protocol are implemented on the security controller. One that is optimized for low memory usage and one that has low computing times at the cost of increased memory consumption. These two versions are then compared against each other to evaluate the performance change if the cryptographic operations for authentication are reduced to a minimum. To demonstrate the capabilities of using different curves and users two curves and users should be supported.

4 Implementation

To test the implementation on the HSM a command structure based on Application Protocol Data Units (APDUs) has been defined. To be able to test the performance of the single steps of the protocol a command for every operation was defined. The operators' device sends the necessary commands to the HSM, retrieves the answers from the HSM, takes timing measurements, and asks the operator for the desired operation and, if necessary for the authentication credentials.

Figure 1 shows the authentication interface on the NFC-enabled smartphone asking the operator for the credentials. This test version is configured to emulate a user and has the demo credentials entered. The implementations support the handling of arbitrary ECC curves; it therefore has a setting to insert the curve parameters manually. The buttons visible can generate the local public key and generate the final key after the communication is complete. After the authentication is finished successfully the operator can view the device's configuration, alter the received configurations, and deploy an altered one.

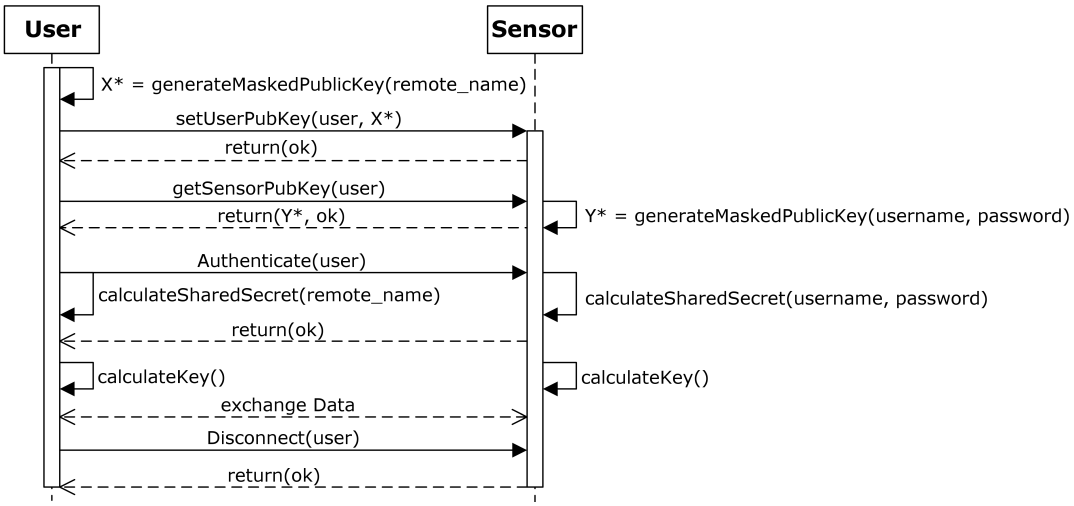

Fig. 2. Communication structure for fine granular evaluation

5 Performance Evaluation

The measurement of the bare communication with different sized payloads is performed with the echo command. If the user is authenticated this command returns the payload, otherwise an empty packet with an error number is returned. As expected, the time used for communication increases linearly with the payload length. Also the time for communication approximately doubles if the payload is also sent back. When the corresponding time is subtracted of the communication time of other commands, it can be evaluated how long the operations on the secure element take to perform.

As expected, the timing of the authenticated echo request is slightly more than double of the unauthenticated one. This is because the data has to be sent back again and some more internal computation has to be done.

In Figure 2 the communication structure for evaluation purposes is shown. It comprises of the calculations on the two sides of the communication and the communication itself.
The average timing of generating the public key on the HSM is about 218800 μs. Considering that the communication of one empty packet and one packet with 64 bytes payload takes on average 8000 μs, one can conclude that the necessary calculations can be performed in 210800 μs or about 210 ms. The key derivation (K_A and K_B) also takes about 200 ms. The calculation of the key from the users credentials and the shared secret uses approximately 37 ms. All those measurements were made 1000 times. The distribution of the values is gaussian. The gaussian parameters for the operations can be found in Table 2. With those numbers we can estimate that the whole authenticated key exchange can be performed in about 426 ms. The time to initialize the users is negligible as only the credentials need to be saved. Other methods like combining ECDH and the protocol proposed in [3] take approximately an equal amount of operations on every run but require additional messages to be sent and perform cryptographic

Table 2. Timings of the different operations

Operation (low-mem)	Mean [μs]	Sigma [μs]
unauthSend(64b)	8023	45
sendKey	8295	61
generatePubKey	218813	205
calculateSharedSecret	203548	142
calculateKey	42381	68
Operation (low-comp)	**Mean [μs]**	**Sigma [μs]**
generatePubKey	74077	67
calculateSharedSecret	67858	125
calculateKey	39554	49
initUser	91002	28
initMachine	12453 + 43542/User	18
changeMachine	273919	170

functions on initializations of new users. A comparison of the necessary operations is shown in Table 3.
If the operations are altered in a way such that more computations are done when initializing a new user (or altering the user-credentials) it would take more memory space but the computation time on every run can be cut down to about 145 ms on every run and 91 ms upon initializing the user. Changing the credentials of the security controller requires that some credentials of the users need to be recalculated. This requires about 110 ms per user. These numbers are shown combined in Table 2 in the *low-comp* section.

Table 3. Comparison between different authentication schemes.

	ECDH	[3]	SPAKE-low-mem	SPAKE-low-comp
Crypto-operations $(*; +; H(\ldots); rand())$	2; 0; 1; 1	3; 2; 6; 1	6; 2; 5; 1	2; 2; 1; 1
initialization crypto-operations	0; 0; 0; 0	3; 2; 0; 0	0; 0; 0; 0	4; 0; 2; 0
Time authentication [ms]	120	279 + ECDH	426	145
Time initialization [ms]	0	203	0	∼100
Permanent Memory / User	0	credentials	credentials	2 Hashes + 2 Points
Permanent Memory SE	0	credentials	credentials	1 Hash

Table 3 compares two possible SPAKE implementations (one designed for low memory usage, one for low computation time) with operations needed when using ECDH and the authentication scheme proposed by [3]. It furthermore shows approximated timings for the operations when executed on the chosen HSM. The protocol from [3] uses a previously secured communication channel

to transmit data. Therefore, an algorithm like ECDH needs to be executed before the authentication is performed. The table shows the required cryptographic operations for every authentication and the operations necessary when initializing a new user. Furthermore, the timings for initializing and authentication on the HSM, and the memory usage for the different algorithms per user and for the HSM are shown. The comparison indicates that the SPAKE protocol is at least as efficient as the protocol proposed by I. Liao, C. Lee and M. Hwang [3]. If the hardware allows for more memory usage, the shown SPAKE2 implementation performs the authentication on top of the ECDH with little overhead.

With these statistics the strength of the implemented SPAKE2 protocol gets visible. While other protocols initially perform a key exchange followed by the authentication, these two steps get done in a single operation. Not only the time used to communicate, but also the time needed to protect the authentication data during transport can be reduced. With this the transmission of data can stop earlier, resulting in decreased energy consumption. Additionally, the amount of required cryptographic operations is reduced. A protocol using an ECDH scheme and an authentication protocol afterwards uses more random numbers, more hash-operations, and more ECC-related functionalities. Furthermore, the memory usage per user can be just a few bytes (the credentials and additional information for authorization and cryptography details). The term *credentials* means that the password and username are stored in plain text inside the tamper resistant memory. In the low-comp section, the credentials are stored in their key-expanded form (*Hash*) as they will be used like this during the computation. Additionally the ECC points M and N are calculated beforehand and stored in the memory to reduce computation time.

6 Future Work

As previously described, the authorization and establishing of the secured channel can be performed with two messages. This can reduce the communication overhead, which is especially useful for low-powered sensors where the use of the communication devices consumes most of the available energy. This protocol can be changed to also enable authentication between machines. This is especially useful for an industrial setting where robots need to communicate with other machinery to fulfil their tasks. It can also be combined with other communication techniques to enhance current sensor configuration possibilities.

7 Conclusion

In this paper an implementation of the SPAKE2-algorithm [6] has been shown. The evaluation shows that the protocol can be implemented efficiently on an HSM. It also shows that the use of an authenticated key agreement protocol is advantageous compared to a standard solution where key agreement and authentication are performed separately. These features naturally reduce the communication overhead and can be implemented with little overhead in computation

or memory size. Combined, this leads to the conclusion that an authenticated key agreement like SPAKE2 is useful for the use on a smart sensor.

Acknowledgment

This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 692480. This Joint Undertaking receives support from the European Unions Horizon 2020 research and innovation programme and Germany, Netherlands, Spain, Austria, Belgium, Slovakia.

References

1. Diffie, W., Hellman, M.: New directions in cryptography. IEEE transactions on Information Theory **22**(6) (1976) 644–654
2. Peyravian, M., Zunic, N.: Methods for protecting password transmission. Computers & Security **19**(5) (2000) 466–469
3. Liao, I.E., Lee, C.C., Hwang, M.S.: A password authentication scheme over insecure networks. Journal of Computer and System Sciences **72**(4) (2006) 727–740
4. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on, IEEE (1992) 72–84
5. Halevi, S., Krawczyk, H.: Public-key cryptography and password protocols. ACM Transactions on Information and System Security (TISSEC) **2**(3) (1999) 230–268
6. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Cryptographers Track at the RSA Conference, Springer (2005) 191–208
7. Bellare, M., Rogaway, P.: The autha protocol for password-based authenticated key exchange. Technical report, Technical report, IEEE (2000)
8. Kobara, K., Imai, H.: Pretty-simple password-authenticated key-exchange under standard assumptions. iacr eprint archieve, 2003
9. Krawczyk, H.: Sigma: The sign-and-macapproach to authenticated diffie-hellman and its use in the ike protocols. In: Annual International Cryptology Conference, Springer (2003) 400–425
10. Gennaro, R., Lindell, Y.: A framework for password-based authenticated key exchange. In: International Conference on the Theory and Applications of Cryptographic Techniques, Springer (2003) 524–543
11. Abdalla, M.: Password-based authenticated key exchange: An overview. In: International Conference on Provable Security, Springer (2014) 1–9
12. Druml, N., Menghin, M., Kuleta, A., Steger, C., Weiss, R., Bock, H., Haid, J.: A flexible and lightweight ecc-based authentication solution for resource constrained systems. In: Digital System Design (DSD), 2014 17th Euromicro Conference on, IEEE (2014) 372–378
13. Ulz, T., Pieber, T., Steger, C., Haas, S., Bock, H., Matischek, R.: Bring your own key for the industrial internet of things. In: Industrial Technology (ICIT), 2017 IEEE International Conference on, IEEE (2017) 1430–1435
14. Google: Google Weave - uWeave. https://weave.googlesource.com/weave/libuweave/+/HEAD (2016)

SystemC Test Case Generation with the Gazebo Simulator

Thomas W. Pieber, Thomas Ulz, and Christian Steger
Institute for Technical Informatics, Graz University of Technology, Inffeldgasse 16/I, Graz, Austria
{*thomas.pieber, thomas.ulz, steger*}*@tugraz.at*

Keywords: SystemC, Gazebo, Plugin, Cosimulation, Simulation, XML, Test Case Generation

Abstract: The current approach of hardware simulators is a testbed, that supplies the *Device under Test (DUT)* with inputs. These sequences of inputs are the result of engineers reverse engineering the use cases extracting the inputs from them and adding some extreme cases. This paper describes an approach where the input sequences are generated directly from the use case itself. The use case is therefore simulated in an environmental simulator such as Gazebo. This generates the stimuli for the DUT. To facilitate the compatibility between the different simulation environments we present an easy-to-use and easy-to-implement communication strategy.

1 INTRODUCTION

A system design needs to be simulated in order to test it extensively. Such a simulation should be stimulated with real world events and unusual events to test the functionality under normal working situations and extreme cases. Such tests are normally designed by thinking of scenarios, defining how the device should react, and testing these input sequences. Furthermore, random input sequences can be applied to the system to test the design more extensively. These random tests are unlikely to produce valid inputs and primarily test the error handling. As these tests are unlikely to perform useful tasks, it is necessary to have a system that can generate useful input and the according expected data. We have therefore designed a system that uses an environment simulation to generate the inputs for testing a sensor system. This design can also reduce the effort needed to design test cases, as only scenarios in which the DUT normally operates need to be constructed. From these scenarios valid input data is automatically generated. As an environmental simulator, a robotics simulator, such as Gazebo, can be used.

A robotics simulator is designed to handle complex systems and generate sensory information of any kind. A widely used robotics simulator is the *Gazebo simulator* (Open Source Robotics Foundation, 2004; Koenig and Howard, 2004). This open source product can be modified (in a, for us, useful way) by writing plugins for the entire world, the models, the sensors, and an entire system. Further modifications can be made to the visuals and the GUI. The Gazebo simulator operates in discrete time steps of 1 ms . This is enough for simulating the movement of a robot and scarce enough that the robot's operating system can handle most commands in this time step.

For simulating a sensory device (hereafter *Device under Test* or *DUT*) a tool like *SystemC* (Accelera, 2000) can be used. With this simulator, a complex microsystem can be designed and tested. Also the component parts of this system can be modelled in varying detail, allowing for later synthetization. SystemC can simulate the DUT in discrete time intervals as small as 1 fs.

This difference in simulation speed is a major problem to be solved to combine the two simulation environments. As the SystemC simulation is operates in such fine time intervals, it generates huge amounts of data during the execution of a test scenario, as such a test scenario can last for many minutes. The problem of generating and handling these amounts of data must be considered when designing the status output of the DUT. Also the connection of the DUT on the SystemC side of the simulation needs to be modified in order to allow the storage of communication data between it and the simulator and the correct input of the data to the DUT.
Another issue that comes with connecting the simulators comes from the communication between them. The simulators have to communicate in order to exchange status information like the actual simulation time, commands, and generated messages. As the SystemC simulation works with finer time steps it re-

Original work published in Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications, pages 67-72. SCITEPRESS - Science and Technology Publications, 2017.

quires much more fine granular input data and produces huge amounts of output data. This data needs to be extrapolated from the data Gazebo provides and afterwards filtered to allow Gazebo to work with the return data.

The remainder of this paper is structured as follows: In Section 2 other works that combine SystemC or Gazebo with other simulators are described. Section 3 explains the motivation for our design, the requirements that need to be implemented, and details of the solution for the requirements. An evaluation of the usability and functionality of the design is described in Section 4. Following that, Section 5 mentions ideas on how to further improve the proposed design. This paper concludes with Section 6.

2 RELATED WORK

Gazebo is an open source robotics simulator. It is primarily combined with the *Robot Operating System (ROS)*. There are some approaches for combining Gazebo with other software for robotics and computational intelligence (Zamora et al., 2016).
There are approaches that connect other tools that can be used to simulate hardware, to Gazebo (Mathworks, 2016). In these approaches, the main interface to the simulation environment is the interface to ROS.

SystemC is a modelling language based on C++. The extension consists of a class library and a simulation kernel. In (Panda, 2001) a short summary of design processes for SystemC is given.

There exist some interfaces to SystemC in the literature such as (Martin et al., 2002; Possadas et al., 2005; Bouchhima et al., 2006; Huang et al., 2008; Mueller-Gritschneder et al., 2013). These papers use SystemC as a primary basis and extend the functionality of it. (Huang et al., 2008) describe a possibility of running a SystemC simulation on a distributed network, improving the time performance significantly.
An interface to SystemC was designed in (Bouchhima et al., 2006). This work uses Matlab/Simulink as a continuous simulation for the environment, which communicates with the SystemC model.
In the work found in (Martin et al., 2002), SystemC was connected with an analog circuit simulator like SPICE (Simulation Program with Integrated Circuit Emphasis) and VHDL (Very High Speed Integrated Circuit Hardware Description Language (VHSIC Hardware Description Language)).
In (Mueller-Gritschneder et al., 2013) a platform for simulating an entire robot is modelled in SystemC. In this approach the SystemC simulation was used to simulate the behaviour of a robotic system on the transaction layer. With their simulation results the authors update the model of a robot in a virtual world, simulated in a Java environment. After that measurements are taken in the Java environment and sent to the SystemC robot model for further processing. They use network sockets to communicate between the two simulations. In this paper SystemC was used to simulate the robot's movement as accurate as possible, while in our proposal the robot's behaviour gives the input to the simulation of other hardware.

In summary, SystemC was connected with many simulators for cycle accurate measurements, other hardware description languages, or circuit simulators. In (Mueller-Gritschneder et al., 2013) SystemC was used to simulate physical effects on robots.

Comparing this previous work with our proposal, our contributions are a completely new aspect of connecting simulators for generating test cases for systems automatically, as well as connecting the SystemC simulation with an open source robotics simulation. That means that SystemC was previously used to simulate effects in the larger simulation, in this approach the larger simulation is used to stimulate the whole system simulated in SystemC.

In our use-case a sensor is read using a wireless channel. For such a use-case the NFC technology is well well-suited. This is due to the fact that energy can be transmitted through the RF (radio frequency) field. With that energy the sensor and the supporting microcontroller(s) can be operated. The same energy can also be used to charge a small battery or capacity. In (Wireless Power Consortium et al., 2010; Strommer et al., 2012; Lee et al., 2013) the mechanism for transmitting energy alongside data and for storing that energy are described.

There are different approaches to communicate between simulation environments; commonly used are XML (Extensible Markup Language) and JSON (JavaScript Object Notation). Of these two, JSON is more efficient as (Nurseitov et al., 2009) show. (Sumaray and Makki, 2012) furthermore compares Google's Protocol Buffer (protobuf), which is used by the Gazebo simulator, alongside JSON and XML.
As the SystemC simulation works in many cycles for every internal time step, the efficient generation of JSON objects, as well as the generation of protobuf-messages, would require major changes in the existing simulations. Following this, the approach to communicate from SystemC to Gazebo is to embed the values of interest in XML-tags.

3 DESIGN

The goal of the presented design is to connect a SystemC simulation to a high-level simulator in order to generate stimuli for the simulation. Using this approach, the testbed for the SystemC simulation is the simulation of the use-case. This method allows the generation of stimuli for the SystemC simulation from the specification of an interaction of the designed system with the environment. This is not only more efficient than a system engineer could be, but also small variations of the environment can generate a wide variety of tests.

To support a SystemC simulation with input from an environment simulator like Gazebo, a plugin for that simulator needs to be developed. This plugin must be able to send commands and data to the SystemC environment and receive the results and status of the SystemC simulation.

A huge hurdle for connecting the two simulation environments is the different simulation speeds. Gazebo works with time steps of 1 ms whilst SystemC can handle steps as small as 1 fs. This is a twelve orders of magnitude higher time resolution of SystemC. Even a "slow" computer which only works at 50 MHz performs $5 \cdot 10^4$ steps in one time step of Gazebo. This results in amounts of data that are hard to evaluate in a high-level simulation environment. Therefore, measures have to be made to limit the amount of data that needs to be transmitted. In order to do this a connection between the simulators needs to be defined.

The connection between the two simulation environments must be supported by both environments. In the Gazebo simulator a plugin can be written. Figure 1 shows a subset of the states the plugin needs to perform during the execution. This model-plugin needs to open communication channels and fork a new process that will become the SystemC simulation. The plugin then proceeds to the normal execution. For that it connects to the required Gazebo internal signals. One of the required connections is to the onUpdate signal. This signal is set on every time step in the simulation. On activation of this signal the plugin retrieves the status information and other messages from the DUT in the SystemC simulation. When this is done and the data is processed, the plugin can send new commands and messages to the DUT.

To send different messages between the communicating parties an easy-to-implement approach is used. In the SystemC simulation the interesting parameters can be sent to the communication channel. The quickest way to implement such a communication would be to redirect the standard output to the communication pipe before forking the SystemC simulation,

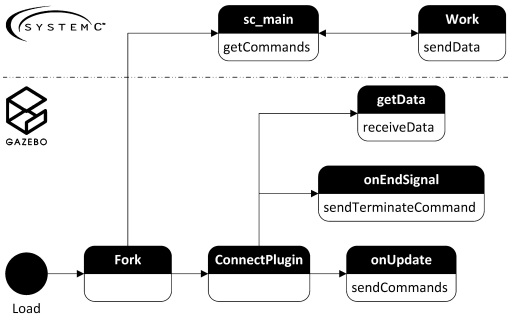

Figure 1: States of the execution of the implemented plugin.

and send the interesting values encased by XML-style tags. With these tags it is easy to find the interesting values in the input stream and separate it from the rest. With this approach the produced data can be collected by Gazebo in the order it was produced. Another approach is the collection of the interesting data in a global data structure, generate a JSON message, and send it at the end of the time step. This approach however would require more changes in the simulation, while the improvement of the efficiency is negligible in contrast to the whole execution.

The Gazebo plugin needs to parse the incoming stream of data. With the XML-tags it can detect the values, perform some preprocessing and store it in a fitting datastructure. When the simulation halts at the specified time, a sync tag needs to be sent to Gazebo. When this signal is received, the plugin can operate again and send the received data to the processing nodes. As (Sumaray and Makki, 2012) show, this communication is done most efficiently with Google's protobuf approach. To use this method, custom messages are defined. These messages can then be sent to a simulation of the channel. This separation is done in order to provide the possibility of changing the channel and the parameters independently of the rest. In our design a NFC communication is chosen. The channel simulation is implemented as a World-plugin of Gazebo. This plugin takes the environmental states and calculates the transmitted power and can induce errors or attacks on the communication. The calculations of the transmission statistics are based on (Wireless Power Consortium et al., 2010; Lee et al., 2013) The channel plugin can then relay the (modified) messages with additional information about the channel to the receiving party. In our scenario, the ROS system can then access the received data.

The communication in the other direction follows the same rules. The robot's OS sends data to the channel, the channel modifies and appends the message and retransmits it to the sensor plugin (if the sensor is within communication range). The sensor plugin

gathers the data and commands from the robot and from the channel, and generates a message that can be sent to the SystemC simulation.
Another useful extension is another World-plugin that gatheres sensory information for the sensor to process. Also this information can be collected by the sensor plugin and relayed to the SystemC simulation.

In the SystemC simulation the testbed needs to be changed to accommodate the interface to the high-level Gazebo simulation. The requirements that the altered testbed has to fulfil, in order to work properly, are:

R.1 The simulation parameters, commands to the DUT, and other information regarding the hardware simulation must be configurable.

R.2 The commands that come from the Gazebo simulator must be parsed and distributed.

R.3 The testbed must be able to be activated continuously.

R.4 The simulation time for each iteration should be variable to allow a variety of scenarios.

R.5 A direct communication to the parts that can be affected by the commands must be possible.

R.6 The incoming commands must be stored and executed in the correct order.

R.7 A proper time synchronization between the two environments needs to be established.

R.8 The traces from the simulation should be deactivated, or at least reduced, as a execution over a prolonged simulation time produces huge amounts of data.

Furthermore, some minor changes have to be made in the rest of the simulation. Most notably is the insertion of messages back to the Gazebo simulator. These changes are similar to the changes needed for requirement R.7.

Requirement R.1 is fulfilled by providing a POSIX (Portable Operating System Interface) pipe that connects the standard input of the SystemC testbed with the Gazebo plugin. This allows the transfer of information (like configuration parameters) through the standard input from the Gazebo simulator to the SystemC simulation.
To satisfy requrement R.2 a parser for the received messages is implemented and called at the start of every execution cycle.
To fulfil requirement R.3 the testbed is written in an endless loop. The condition to terminate the process is taken from the standard input. As the input is connected via a POSIX pipe to the Gazebo simulator, the SystemC simulation can be ended if Gazebo is closed.
Special commands to change timing parameters are included in the command structure to implement requirement R.4. In SystemC two commands are needed; one for the time unit and one for the value.
As we want to simulate a sensor system, it is useful to feed the simulated sensory information to the sensor-input of the SystemC simulation. This corresponds to specification R.5. With special tags this data can be extracted from the received message. This data is then sent to a FIFO (first in, first out) memory "sensor". This sensor can then set the required values on its data lines.
The same approach can be used for requirement R.6. In this requirement the input is stored in a FIFO at the control unit of the sensor. This control unit can then execute the commands in order. For a proper execution of the commands an additional data field is required. This data field stores the exact time values at which the command should have arrived to simulate a serialized channel.
To fulfil requirement R.7, a channel to communicate back to the Gazebo simulation needs to be declared. This is most easily done by redirecting the standard output to another pipe before executing the simulation. The synchronization between the simulation environments is done by creating a sync-signal at the end of a simulation step. The sync-signal is defined by a specialized XML-tag that is sent to the standard output. After this signal is received, the Gazebo simulator is allowed to excite another step. This step in turn triggers the advance of the SystemC simulation by one time step. The same method for communicating to the Gazebo environment can be used for other data as well. These data-fragments are then encased in XML-tags that can be found in the output stream of the SystemC simulation. For each interesting event in the SystemC environment a separate XML-tag is defined. In our test-design, interesting events are: (I) the sending of data packets from different execution stages and (II) the status of the battery of the sensor.
When simulating an environment it is necessary that longer amounts of time are simulated, in order to also simulate the edge conditions. This results in huge amounts of data that come from the SystemC environment. Normally, this data comes in form of VCD-files (Value Change Dump files). If the clock of the simulation is dumped as well, this results in vast amounts of data referring to clock changes while the actual simulation process has not even begun.
Requirement R.8 refers to that problem. Therefore it is necessary to wait for input before initializing the simulation. This input determines if traces will be made, and if yes, which traces should be activated. Furthermore, it is possible to pause the execution of the SystemC simulation if all relevant operations for

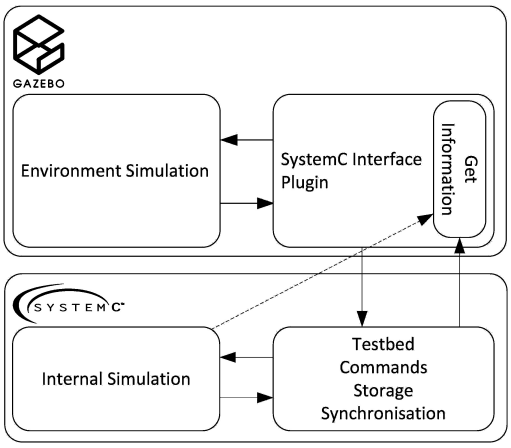

Figure 2: Overall communication process between Gazebo and SystemC.

this time step are handeled which results in shorter simulation times. This approach does however reduce the quality of the simulation results and should therefore only be used if the DUT is in a low power mode.

If the simulation is paused prematurely two additional challenges emerge. The first is the disconnection of the simulation time of Gazebo and SystemC. In order to prevent that from happening in the trace, an additional value can be generated. This value is toggled if the simulation pauses prematurely.
The second challenge is the simulation of values that change even if basically no useful operation is performed. An example for such a value is the energy level of a battery that is drained a little even in low-power mode. A fashionable solution to this problem is the estimation of this value at the supposed end of the simulation.
With these calculations at the premature pausing of the simulation and the knowledge that every simulation step takes 1 ms the timings of the operations can be restored and the sharp edges on estimated values can be reduced by interpolating between the data points.

Figure 2 shows the global communication paths between the two simulation systems and the changed interfaces. The robot from the Gazebo simulator, as well as the environment in which the robot operates, compute the input for the SystemC simulation. These input parameters are sent to the interface (testbed) of the SystemC simulation, which in turn distributes this data to the required parts of the internal simulation. This simulation produces outputs which are sent over the standard output to the Gazebo simulator (dashed line). Furthermore, data is sent to the testbed, as some traces still need to be captured. In the Gazebo simulator, the environment can affect the communication channel between the sensor and the robot again. Finally, the robot can access the gathered data. This data can then be compared to the data generated by the sensor in order to optimize the communication process.

All parts of the simulation in the Gazebo simulator are implemented using plugins. This allows the easy reconfiguration of the simulation to feature other or more actors, inputs, obstacles, parameters, and even allows the change of the whole communication channel.

4 EVALUATION

To evaluate the developed system, a SystemC simulation of a smart sensor that communicates and charges the internal battery with NFC (Near Field Communication) technology was constructed. This is the sensory device denoted DUT beforehand. This simulation is then started by a Gazebo plugin. The Gazebo simulator provides the context for the simulation and gives useful sensory values as input to the system. The Gazebo simulator can communicate with the SystemC simulation over the standard input and output of the SystemC simulation. In our design the wireless NFC communication channel is modelled in a Gazebo World-plugin. A World-plugin is a plugin, that is loaded at the beginning, connected to all parts of the simulation, and is active until the simulation ends. With this approach, the channel simulation can get different parameters that affect the communication. Such parameters are the distance and orientation between the antennas, obstacles between the antennas, and other noise on the communication band. Because the channel is modelled separately, it can be easily changed to accommodate other communication techniques like WiFi, Bluetooth, Zigbee, or wired connections.

The communication between the simulators is performed with strings encased in XML-tags. On each side of the communication an XML parser is written that searches for the declared tags. The strings found by this parser are then stored in a suiting data structure. For our use case, this data structure consists of a list of objects, referring to the XML-tags, each containing a list of the received data values.
The data contained in this structure can then be read sorted by topic in the order it was received. For data that needs to be transmitted via the NFC channel, a preprocessing step is executed that generates protobuf-messages.

The SystemC simulation can get the required stimuli from the environment of the Gazebo simulator. In Figure 3 a concept for the simulation is shown.

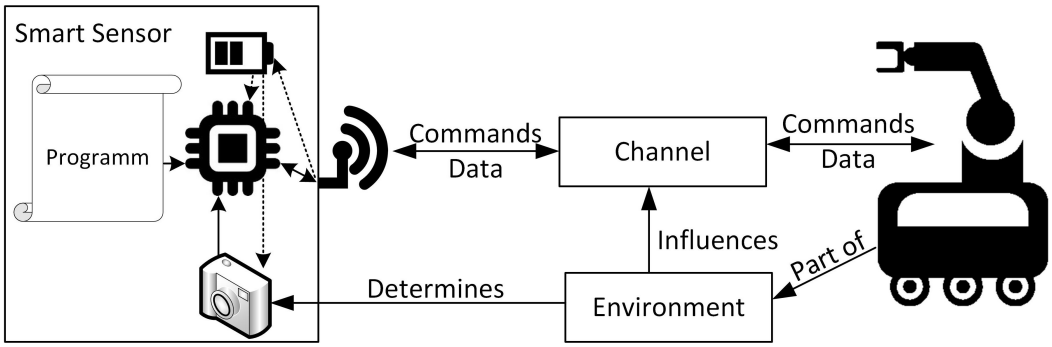

Figure 3: Concept of the Evaluation Design.

The robot, as part of the environment, interacts with the DUT. In this simulation the DUT is attached to a crate as an environmental feature. The robot moves towards the DUT and tries to communicate with it. As the robot gets closer, the NFC connection between the robot and the DUT gets better. When the channel is good enough such that enough energy gets transmitted the DUT switches on and the communication can happen.

This simulation performs every operational phase of such a DUT when some entity tries to communicate with it. Because the global simulation is done in a robotics simulator, a new test case can be implemented by repositioning the DUT and robot and giving the robot some new instructions. The local simulation of the DUT is still done with SystemC. This allows very accurate measurements on the DUT. In this scenario we want to monitor the power usage and ability to store excess energy during such a read cycle. With these measurements we hope to develop a more efficient power manage system on our DUT. Additionally more information about the NFC communication can be gathered from this simulation.

Figure 4 shows the environment with the robotic arm and the DUT. To communicate with the DUT another sensor was mounted on the end of the robotic arm.

While the robot is approaching the sensor the NFC antenna on the robot's arm is activated. The transmitted energy and commands are received at the sensor. The sensor's energy intake can be seen in Figure 5. This figure furthermore shows the compression of simulation time as a result of requirement R.8. The upper graph starts at approximately 0.9 ms and runs to about 3.5 ms. The lower graph is time-expanded to correlate with the Gazebo simulation time. This graph starts at 5840 ms and runs to 6480 ms. This results in a compression of the 5.8 s idle time to 0.9 ms.

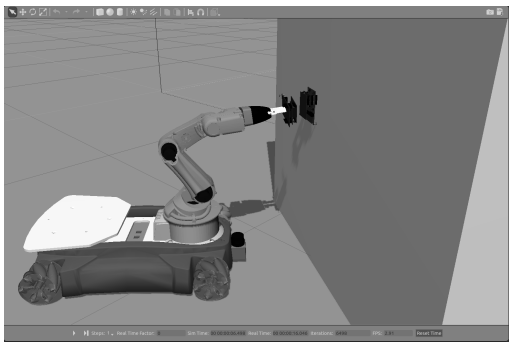

Figure 4: Interaction of a robot with the sensor.

5 FUTURE WORK

As many hardware elements are simulated in languages other than SystemC, plugin-structures need to be developed to also include such simulations with a high-level simulator. From our point of view a similar approach to include languages such as Verilog or VHDL seems promising.
For bigger simulations it may also be useful to connect to the SystemC simulation using network sockets to allow the parallel computation of multiple sensors. Furthermore, simulations of the planned system as a whole, of multiple sensors, senders and receivers, and different channels need to be performed.

6 CONCLUSIONS

We presented a new approach for connecting a SystemC hardware simulation to a robotic simulator. This is done in order to automatically generate the stimuli for the SystemC testbed. The huge difference in time resolution between these simulators pose a barrier to connect them. To solve this hurdle, a mechanism to synchronize the simulations and

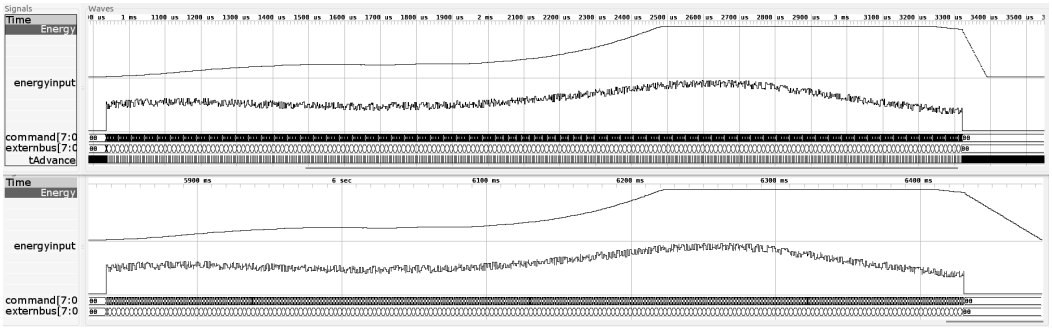

Figure 5: Energy trace of the sensor while a robot is approaching. The shorter simulation time at the top is a result of the reduction of execution time due to requirement R.8

transport data between them is shown. In addition to that, we have mentioned that extrapolating data for the SystemC simulation, as well as filtering the produced data for the Gazebo simulator, is necessary. Furthermore, the SystemC simulation needs some changes in order to be able to connect to such a high-level simulator. These changes include:

- *Reduction of output* if possible - without greatly decreasing the degree of detail where it is needed.
- *Rewriting the testbed* to allow the external simulation to control the simulation
- *Inclusion of special outputs* for synchronization, passing status information, and transmitting the generated data to the Gazebo simulator.

Additional changes have been made to the Gazebo simulator.

- *A plugin to control SystemC* has been developed. This plugin handles all communication from and to SystemC, keeps the simulations synchronized, and controls the start and end conditions for the SystemC simulation.
- *Different channels*, wireless and wired, have been modelled in Gazebo that can be used to simulate the connection between arbitrary devices.

ACKNOWLEDGEMENTS

This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 692480. This Joint Undertaking receives support from the European Unions Horizon 2020 research and innovation programme and Germany, Netherlands, Spain, Austria, Belgium, Slovakia.

REFERENCES

Accelera (2000). SystemC. http://accellera.org/downloads/standards/systemc. Last accessed on Jan 17, 2017.

Bouchhima, F., Briere, M., Nicolescu, G., Abid, M., and Aboulhamid, E. (2006). A SystemC/simulink co-simulation framework for continuous/discrete-events simulation. In *2006 IEEE International Behavioral Modeling and Simulation Workshop*. Institute of Electrical and Electronics Engineers (IEEE).

Huang, K., Bacivarov, I., Hugelshofer, F., and Thiele, L. (2008). Scalably distributed SystemC simulation for embedded applications. In *2008 International Symposium on Industrial Embedded Systems*. Institute of Electrical and Electronics Engineers (IEEE).

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Institute of Electrical and Electronics Engineers (IEEE).

Lee, W.-S., Son, W.-I., Oh, K.-S., and Yu, J.-W. (2013). Contactless energy transfer systems using antiparallel resonant loops. *IEEE Transactions on Industrial Electronics*, 60(1):350–359.

Martin, D., Wilsey, P., Hoekstra, R., Keiter, E., Hutchinson, S., Russo, T., and Waters, L. (2002). Integrating multiple parallel simulation engines for mixed-technology parallel simulation. In *Proceedings 35th Annual Simulation Symposium. SS 2002*. Institute of Electrical and Electronics Engineers (IEEE).

Mathworks (2016). Get Started with Gazebo and a Simulated TurtleBot. https://de.mathworks.com/help/robotics/examples/get-started-with-gazebo-and-a-simulated-turtlebot.html. Last accessed on Jan 03, 2017.

Mueller-Gritschneder, D., Lu, K., Wallander, E., Greim, M., and Schlichtmann, U. (2013). A virtual prototyping platform for real-time systems with a case study for a two-wheeled robot. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. EDAA.

Nurseitov, N., Paulson, M., Reynolds, R., and Izurieta, C. (2009). Comparison of json and xml data interchange formats: A case study. *Caine*, 2009:157–162.

Open Source Robotics Foundation (2004). Gazebo simulator. `http://www.gazebosim.org`. Last accessed on Jan 03, 2017.

Panda, P. R. (2001). SystemC - Amodelling platform supporting multiple design abstractions. In *Proceedings of the 14th international symposium on Systems synthesis - ISSS*. Association for Computing Machinery (ACM).

Possadas, H., Adamez, J. A., Villar, E., Blasco, F., and Escuder, F. (2005). RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model. *Design Automation for Embedded Systems*.

Strommer, E., Jurvansuu, M., Tuikka, T., Ylisaukko-oja, A., Rapakko, H., and Vesterinen, J. (2012). NFC-enabled wireless charging. In *2012 4th International Workshop on Near Field Communication*. Institute of Electrical and Electronics Engineers (IEEE).

Sumaray, A. and Makki, S. K. (2012). A comparison of data serialization formats for optimal efficiency on a mobile platform. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '12, pages 48:1–48:6, New York, NY, USA. ACM.

Wireless Power Consortium et al. (2010). System description wireless power transfer. *Volume I: Low Power, Part*, 1.

Zamora, I., Lopez, N. G., Vilches, V. M., and Cordero, A. H. (2016). Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo. *arXiv preprint arXiv:1608.05742*.

Using Gazebo to Generate Use Case Based Stimuli for SystemC

Thomas W. Pieber, Thomas Ulz, and Christian Steger

Graz University of Technology - Institute for Technical Informatics, Graz, Austria
{thomas.pieber, thomas.ulz, steger}@tugraz.at

Abstract. Realistic simulations of new hardware are of utmost importance to achieve good results. The current approach to such simulations is that the Device under Test is exposed to stimuli that are either generated randomly, or that are generated by engineers reverse engineering the use cases and extending the inputs by some extreme cases. In this paper we describe an approach to generate useful stimuli for a SystemC simulation directly from a simulation of the use case. In this approach the use case is simulated using the Gazebo simulator. The stimuli for the Device under Test are then extracted and sent to the SystemC simulation.

1 Introduction

In the development of new systems simulations need to be performed to find errors early. With such simulations the developed system (or Device Under Test "DUT") can be tested extensively and optimizations and error corrections can be implemented quickly and inexpensively. These simulations are usually stimulated with events that occur in the expected use case as well as some extreme cases. These tests are designed by engineers reworking the scenarios and defining the inputs and the expected behaviour. In addition to these tests, random input sequences can be applied to test the DUT's reactions to faulty or unexpected inputs as random input is unlikely to be valid. All together that means that the current test procedure consists of valid inputs designed by the system engineers and (mostly) invalid inputs generated by random testing. We therefore propose an architecture for a generator that can produce valid inputs to the DUT design which can also be evaluated according to the expectations of the engineers. Such system can decrease the effort needed to design tests for the DUT, as only the valid scenarios need to be described. These will then automatically generate valid input data and the expected output.

For such simulation the environment in which the DUT should operate can be simulated. This environmental description only needs to describe the essential parameters that can affect the DUT. Such simulations can be performed in a simulator such as the Gazebo simulator [6, 13]. This simulator is designed for robotic use cases and is designed to handle complex systems and generate accurate sensor information of any kind. This open source simulator also allows modifications to be as useful and accurate as we want it to be. These modifications are done by implementing plugins for the environment (world), the models,

Original work published in Advances in Intelligent Systems and Computing, pages 241-256. Springer International Publishing, 2018.

the sensors, the simulation core, the visuals, and the GUI. This simulator operates in discrete time steps of 1 ms. This degree of simulation accuracy is enough to simulate the movement of robots and sparse enough that the robot's operating system can handle most commands in this time step.

To simulate our DUT another tool such as SystemC [1] can be used. With this tool a complex microsystem can be designed and tested. Furthermore, the component parts of the system can be modelled in various degrees of detail. This allows for accurate simulations or even synthetization of the newly developed parts and efficient simulation of existing hardware. SystemC operates in discrete time intervals as small as 1 fs.

When combining these two simulations, this difference in simulation speed poses a major problem. The execution of a test scenario can last for many minutes. In combination with the fine grained simulation time steps of SystemC this can generate huge amounts of data which need to be handled. This problem needs to be considered when choosing the traced signals and information that should be transferred between the simulations. Additionally, the testbench of the SystemC simulation must be altered to include the communication between the simulations.
This leads to the issue of the communication itself. The simulations need to exchange data such as the generated input and output of the simulation step, as well as status information about the simulations itself. The difference in time steps also introduces the problem that SystemC requires data in more detail from the Gazebo simulation. This data is to be estimated and extrapolated from the existing inputs. It also produces output data that is filtered to allow Gazebo to work with the resulting data.

This paper is based on the work done by Pieber et al. [15]. It expands the ideas behind that publication, gives more detail on the design and implementation of the combination of the simulations. It furthermore expands the evaluation by constructing a detailed simulation run and interpreting the results.

The remainder of this paper is structured as follows: In Section 2 other works that combine SystemC or Gazebo with other simulators are described. Section 3 explains the motivation for our design, states the requirements that need to be implemented, and gives details on the solution for the requirements. An evaluation of the design is described in Section 4. This is done by constructing and analyzing a sample simulation run. Following that, Section 5 mentions ideas on how to further improve the proposed design. This paper concludes with Section 6.

2 Related Work

As Gazebo is an open source simulator for robotics it is primarily combined with a robot operating system such as ROS [19] or YARP [9] to control the simulated robots [10].
There are approaches to combine the Gazebo simulator to software for robotics and computational intelligence [21]. There are further works that connect other

tools that can simulate hardware [8], but the main approach in these works is to use the interface from Gazebo to ROS and implement ROS nodes to connect to the rest.

A design process for SystemC is given by Panda [14]. With this language a model for complex systems can be described and executed. There are many publications that implement interfaces to SystemC as it provides a good basis for simulations [3, 4, 11, 16]. In these approaches the functionality of SystemC is extended to provide the functions needed by the researchers.
An interface from SystemC to Matlab/Simulink was designed by Bouchhima et al. Here the SystemC simulation was stimulated by a continuous environment simulation written in Matlab/Simulink. SystemC was also connected to analog circuit simulators like SPICE [5] and VHDL [2] to improve on flexibility and simulation performance.
Mueller-Gritschneder et al. developed a robot simulation platform in SystemC [11]. They simulate the behavior of the robot on the transaction layer and forward the results to an environment simulation written in Java. They do this in order to simulate the movement of the robot as accurate as possible. In this paper the robot is simulated in SystemC, while in this proposal the robot's behavior is the input to the SystemC simulation to simulate parts of the environment.

In summary SystemC was connected to many other simulations. It is then used as core for other simulations or to generate more accurate results. In the context of robotics, SystemC has been used to simulate the movement of the robot. In contrast to that, this publication uses the robotic simulation to stimulate SystemC components with inputs from the environment to automatically generate valid stimuli.

In this paper, a use-case is evaluated where a sensor measures data from the environment, and is read out and charged via Near Field Communication (NFC) by a robot. Some publications describe the techniques used to transmit data alongside energy and storing the excess energy in small batteries or capacitors [7, 17, 20].

To connect simulations a common interface must be created over which data can be exchanged. In this approach the common interface used is a POSIX (Portable Operating System Interface) pipe where XML (Extensible Markup Language) formatted data is sent. Another possibility to format data efficiently is the JSON (JavaScript Object Notation) format. This would be more efficient than XML [12], but due to other reasons, explained below, the XML format is chosen. Gazebo uses Googles Protocol Buffer (ProtoBuf) as formatting method to transmit data internally. Sumaray and Makki compare the efficiency of this protocol to XML and JSON [18].

Based on [15], this publication expands on the detail of the design and implementation of the developed connection of the simulations. Main focus of the expansion is on details concerning the Gazebo plugin. Additionally the evaluation of the system is expanded. Here, detailed information on the traces produced by SystemC is given.

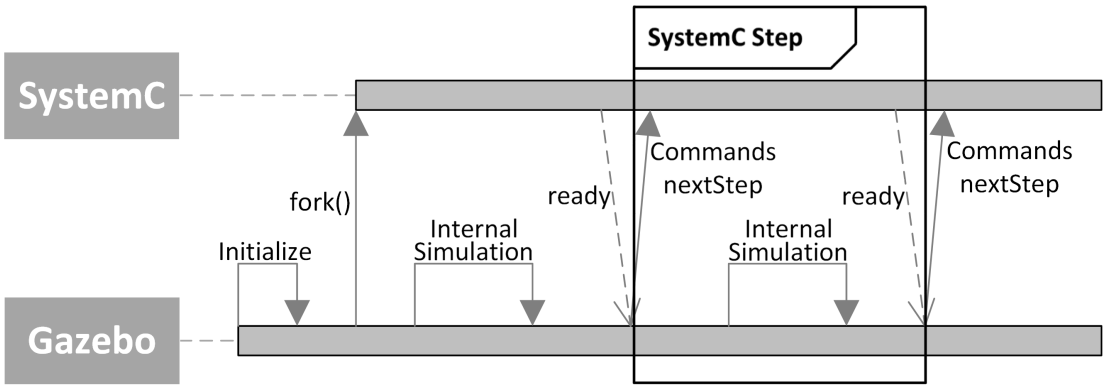

Fig. 1. States of the execution of the implemented plugin.

3 Design and Implementation

The goal of the presented design is to find a method to generate stimuli for a SystemC simulation automatically and being able to see how the simulated system behaves in the specific use-case. To enable this, a connection between SystemC and a high-level simulation is established. This high-level simulation (in this approach the Gazebo simulator) represents the surroundings of the newly developed system (a sensor in this use-case). Using the data from the Gazebo simulator, stimuli for the sensor can be created. That means that the environment of the sensor becomes the de-facto testbed. With this method the stimuli for the SystemC simulation are generated by the interaction of the sensor with the environment. This generates the stimuli not only faster than an engineer could, but also only small variations in the environment can generate a wide variety of different test scenarios such as more noise in the communication or energy fluctuations due to movements of the reader or changed mutual inductance due to small changes in the distance between the antennas.

To connect two simulations successful, both must support the interfaces necessary. To do so, an overall structure for the communication was developed. This structure is shown in Figure 1. This plan visualizes how the simulations are connected, how they communicate and when operations are performed. This figure also shows the minimum requirements that this approach needs to work. In this example the SystemC process is forked from the Gazebo simulation. Here some initial configuration concerning the communication can be set up. One step of the Gazebo simulation then invokes one from SystemC. A return message from SystemC informs Gazebo that the simulation step of SystemC is properly executed. During the execution of the SystemC step additional messages for Gazebo may be sent that need to be captured from the Gazebo environment.

Gazebo can be extended by the use of plugins. To apply the input of the Gazebo environment to a SystemC simulation, one such plugin that handles the communication needs to be developed. The structure for that plugin can be seen in Figure 2. This structure implements the following five operations.

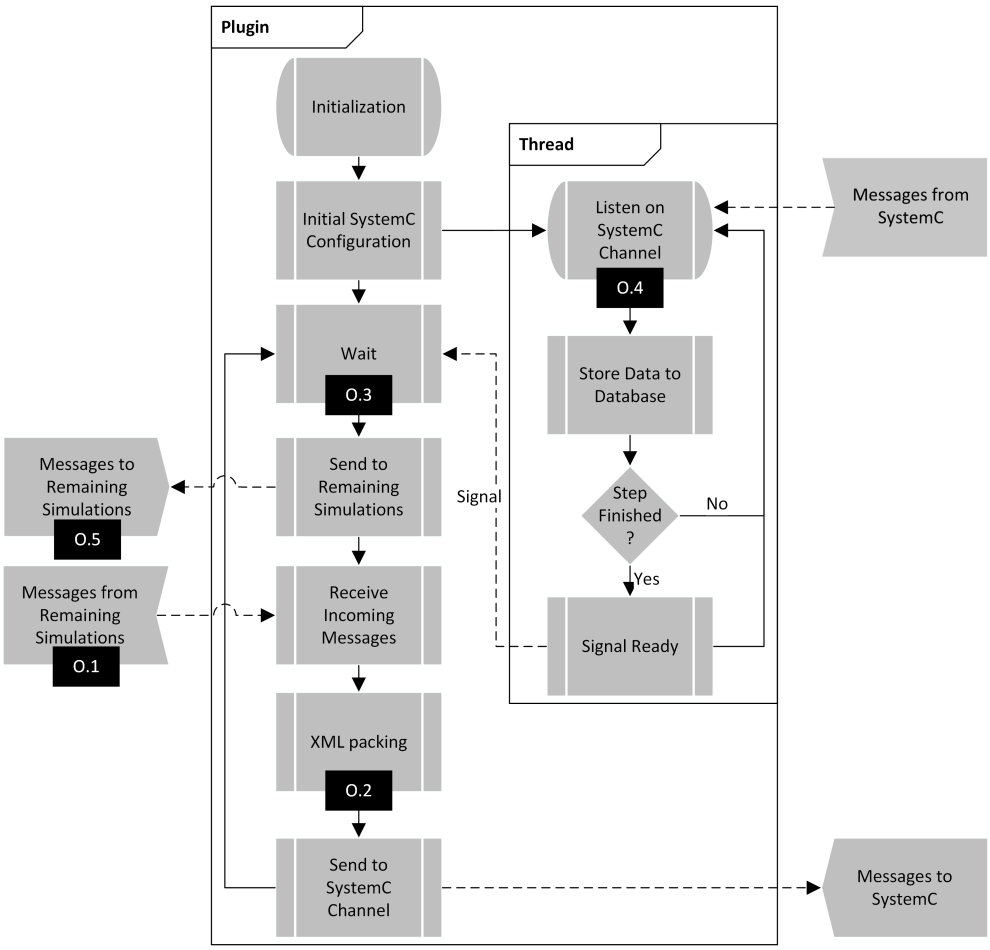

Fig. 2. States of the execution of the implemented plugin.

O.1 The required data must be collected from the environment. That includes data that the sensor can measure as well as communication data.
O.2 The collected data needs to be packed into messages that can be sent to the SystemC simulation.
O.3 The plugin needs to halt the simulation of the environment until the SystemC simulation step finishes.
O.4 During the execution of the SystemC step, all messages needed for the remaining simulation(s) need to be received, stored, and ordered.
O.5 The collected information needs to be distributed to the remaining simulation(s). This can be communication data, visual data, or status information.

The operation defined in O.1 is needed to generate valid input to the sensor that can be evaluated. To generate this information a world plugin may be needed to gather the information. This operation can then be achieved by generating a Gazebo internal communication from this plugin to the plugin connecting the SystemC simulation. Additionally a communication path between the communicating entities must be established. This communication may also be altered in order to simulate the effect of the channel.

To send the gathered information to SystemC operation O.2 is required. This operation formats the data in a way suitable for transport to SystemC. To send arbitrary data the data items are converted into string format and packed using an XML structure.
To properly synchronize the two simulations Gazebo needs to be able to wait for SystemC to finish calculating the current time step. That implies that the plugin needs to be able to halt the Gazebo simulation until it receives the signal indicating the completion. This operation is referred to as O.3.
While Gazebo is waiting for SystemC to finish, SystemC may send different messages that are needed for the rest of the Gazebo simulation. This information needs to be processed and stored until Gazebo can simulate the next time step. When this happens, the plugin needs to forward the information received from SystemC to the rest of the simulation. This operation is described in O.4.
Operation O.5 describes the correct distribution of the gathered data. As SystemC can send different data (e.g.: visual updates such as LEDs, or data for communicating with other entities), the information needs to be split into the topics and sent to their destination using an internal communication mechanism.

The difference in simulation speeds is one of the biggest hurdles in connecting the two simulations. As a tool for simulating interactions and movements of robots, Gazebo works with time steps of 1 ms. On the other hand, SystemC can handle steps as small as 1 fs. This is needed for simulating hardware components as also a "slow" computer which only works with 50 MHz performs $5 \cdot 10^4$ operations in one time step of Gazebo. This difference of twelve orders of magnitude of the simulations can result in massive amounts of data generated by SystemC which is hard to evaluate in Gazebo. Therefore, some measures to limit the amounts of data that are transferred need to be implemented. This can be done best when defining the requirements for the connection between the simulators.

As the two simulations must be compatible in their interfaces to each other, the structure of a SystemC simulation needs to be adopted. Figure 3 shows the overall structure of such SystemC simulation. The messages coming from the Gazebo simulator are received and analyzed. If some parameters need changing, the adaptations are done. To reduce the simulation time, it is evaluated if the changes require instant action. Should that not be the case, the time that should be simulated is added as a debt in comparison to the Gazebo simulation. An action is required if the time debt is too large or if the sensor receives messages that require an answer. To simulate these actions in the correct order, the old parameters and commands are reset and the simulation is run to balance the time debt. In this run the conditions at the current time instance are estimated. After that the new parameters are set and the simulation is started for this time step. When a stable condition is reached the simulation is halted and the conditions for the end of the step is calculated. Should the calculations need more time than one time step, the checking if action is required evaluates to "yes" in the next time step.

To send arbitrary messages between the two simulation environments, an easy-to-implement approach is used. As the SystemC simulation gets forked from

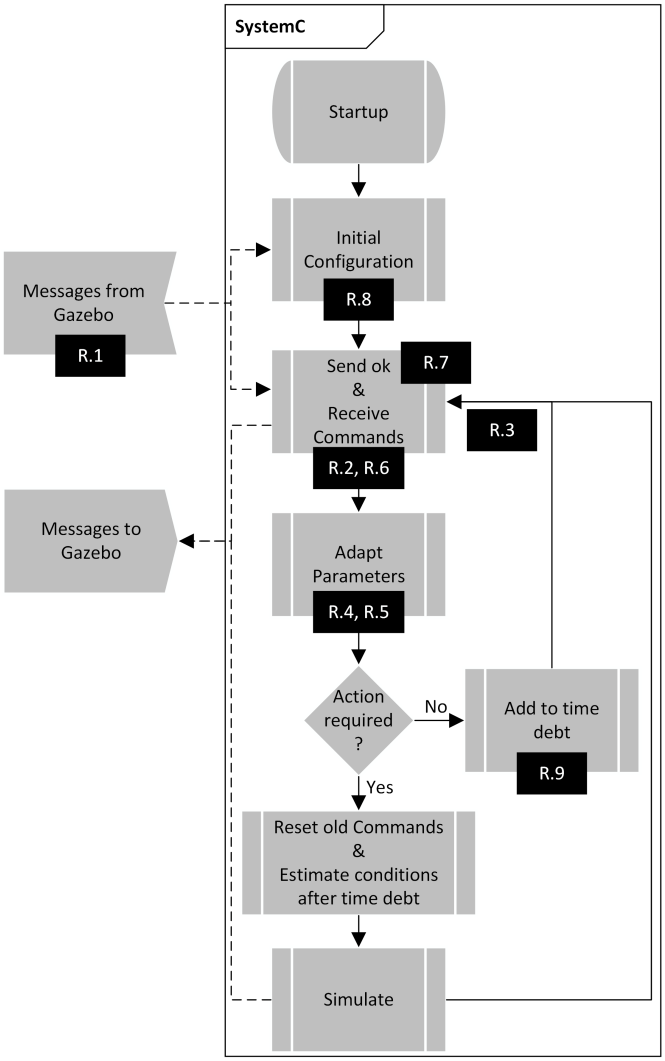

Fig. 3. Structure of the SystemC simulation.

the Gazebo plugin (Figure 1) the standard input and output can be redirected to a POSIX pipe. The Gazebo plugin uses that pipe to exchange messages with the SystemC simulation. This interface allows the transmission of string-type messages. So the commands and parameters need to be packed in a format that can be sent as string and the string received needs to be parsed in order to get the information back. An easy to implement method is the use of XML (Extensible Markup Language) data structure to pack the information in strings. That means that the interesting values are encased in XML-style tags. With these tags the string can be split in parts containing different types of data, which are then evaluated. The use of XML also allows the SystemC components themselves to send their information as soon as it is available and Gazebo can receive and process the data in the order it was produced. Although JSON would be more efficient than XML, XML was chosen because this makes the composing of messages inside SystemC more easy. With XML the messages can be sent

whenever the information is available, with JSON on the other hand the Information needs to be packed in smaller JSON objects which can be sent. This can introduce additional computational overhead, leading to JSON loose its better performance.
The data sent by the SystemC simulation is received by the Gazebo plugin and parsed. The incoming data stream is split into data chunks according to the XML-tags, preprocessed and stored in a fitting datastructure. When the SystemC simulation halts for the time step, a tag for synchronizing the simulations is sent to Gazebo. The reception of this tag signalizes the Gazebo plugin to transmit the collected data to the rest of the simulation and resume its work.
Gazebo is capable of using many types of communication, but Google's Protocol Buffer (Protobuf) is the most efficient communication it supports [18]. To fully use this method, custom messages can be defined that can hold various types of data. The messages from the sensor that are intended for the robot (in this scenario) are for example transmitted to a simulation of the transmission channel. There environmental information is used to simulate signal degradation due to free space loss and multi-path interference. The channel is simulated separately from the rest to be able to test different communication channels such as WiFi, Bluetooth, Zigbee, or in this case NFC. The calculations of the transmission statistics are based on [7, 20]. The channel then modifies the transmitted data and forwards it to the robot, where it can be accessed.
Communication from the robot to the sensor follows the same rules. The robot sends the data to the channel simulation. There, errors are introduced and the signal strength and received energy is calculated. This information is then sent to the sensor plugin which also collects the data for the sensor system itself. The collected data is packed in an XML message and sent to the SystemC simulation. The SystemC simulation receives the commands, parses the XML data and performs the actions needed to simulate the sensor accurately. To accomplish this, the original testbed must be modified to accommodate the interface to the Gazebo simulation. In order for that to work properly, the following requirements need to be fulfilled:

R.1 Gazebo must be able to transfer information about the simulation, commands and parameters for the DUT to SystemC.
R.2 To be able to react to changes in the environment, the SystemC simulation needs to operate in steps. Between the steps the information exchange can occur.
R.3 The SystemC interface for the communication must be able to parse and distribute the received information.
R.4 To support different types of simulation, the simulation time step of SystemC should be variable at each step.
R.5 Parts of the SystemC simulation that need special information need to be able to get it directly.
R.6 Commands received from the Gazebo simulator need to be executed in the order they arrive.
R.7 The two simulations need to be synchronized during the execution.

R.8 To reduce the memory required to run the simulation for extended periods, traces from the simulation should be able to be deactivated.

(R.9 The simulation should be done as quickly as possible, while maintaining the accuracy where needed.)

In addition to the changes in the testbed - now the interface to the Gazebo simulation - some adaptations in the rest of the simulation have been made. The most notable adaptation is the insertion of messages that are sent to the Gazebo simulator. These changes need to be made as the bulk of information is evaluated during the simulation by the Gazebo simulator instead of after the simulation. These changes are made in a similar fashion as the changes needed for requirement R.7.
Figure 3 additionally shows the parts of the SystemC simulation where the implementations of the requirements are mostly located.

To be able to fulfill requirement R.1 a POSIX pipe can be used. This allows the transport of information in string format between SystemC and Gazebo. For that, the standard input and output of the SystemC simulation are rerouted.
An XML parser is needed to split the gathered information into the data chunks needed for the different settings and commands. With the use of a global datastructure the distribution to all parts of the simulation can be performed. These measures fulfill requirement R.3.
The call of the start procedure for the SystemC simulation ("sc_start(...)") is inside a loop. The loop is halted when the simulation waits for input from the Gazebo simulator and the condition to exit the loop is sent from Gazebo when exiting or resetting the Gazebo simulation.
To allow a multitude of simulations the time step size of SystemC may need to be changed during runtime. This is represented in requirement R.4. To meet this requirement two special commands are added. One to set the time unit and one to set the numerical value of the time step.
To support requirement R.5 we modified not only the testbed, but also parts of the simulation. For every module we want a direct communication path to, we need a special tag to extract the data. This data is then written into a global datastructure to be accessible by all modules. The interface from the module itself retrieves the data from this datastructure and can perform the needed operations without the need to communicate the information through the layers of the module. The finished data is then stored in FIFO (fist in, first out) structures to wait for the simulation to need it. This requirement is especially useful for this scenario as we modeled a sensor system and the data the sensor measures is represented in the Gazebo simulation.
Requirement R.6 can be met with the same strategy. This time the FIFO buffer storage is located in the control unit of the sensor. Now the commands that have arrived can then be loaded and executed in the order they should have arrived. To properly execute the commands an additional field that stores the time when the command should have arrived is needed to simulate a serialized channel.
To get a proper time synchronization (R.7) the SystemC simulation needs to send a marker message back to Gazebo indicating that the step has been pro-

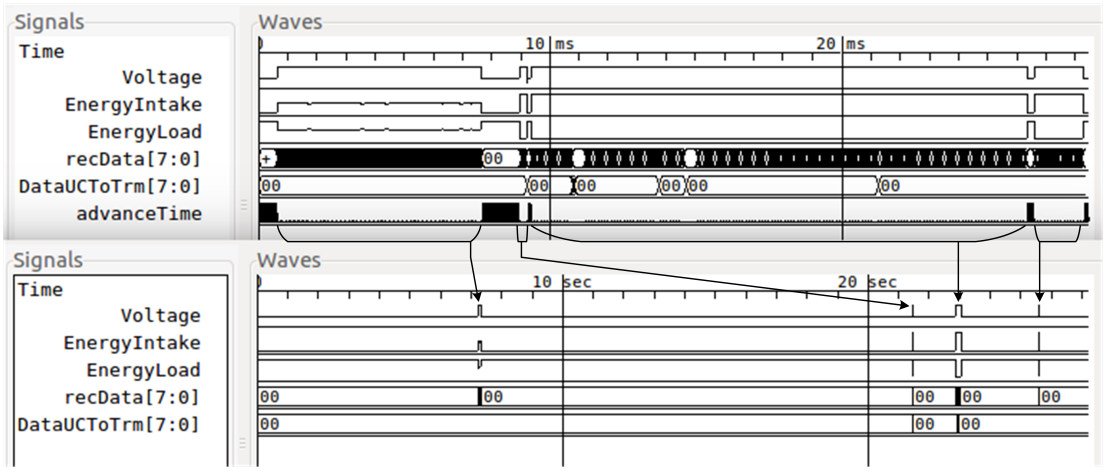

Fig. 4. Compression of idle-time.

cessed. This is done by declaring an extra XML-tag. After receiving this signal the Gazebo simulation is allowed to execute another step - again triggering the SystemC simulation.
To use a simulated environment as generator for system stimuli it is necessary to simulate longer periods of time before the system is initially triggered. This is done in order to generate different edge conditions on the simulated device. As we do not know what exactly the sensor measures during this time, it is necessary to also simulate this time in SystemC. Furthermore, the environment simulations can be run for an extended period of time between stimuli. When storing all generated data large files are created. Requirement R.8 refers to that problem. This can be solved by activating only traces that are needed. The information which traces are needed can be received during the initialization process. This information does not only contain which traces are needed, but also if the traces should be active at all, and what the file name should be.
To further decrease the need for memory, a detector system is be implemented that determines weather a simulation step can be stopped prematurely, or even needs to be started. This detector has the potential to reduce the simulation time significantly and corresponds to the optional requirement R.9.

Such detector can only be implemented if detailed knowledge of the inner workings of the simulation and the system it simulates is available. This also requires some major appendices to the existing simulation.

When pausing the simulation prematurely, two challenges emerge. The first one is the desynchronization of the two simulations. As SystemC offers no methods to change the simulation time when it is stopped, we introduced flag signals that get triggered if the simulation time should get changed. With the help of these signals and some post processing of the generated traces the synchronization can be restored afterwards. Figure 4 shows a trace before (above) and after the decompression of idle-times is performed. The markers between the traces indicate the compression.

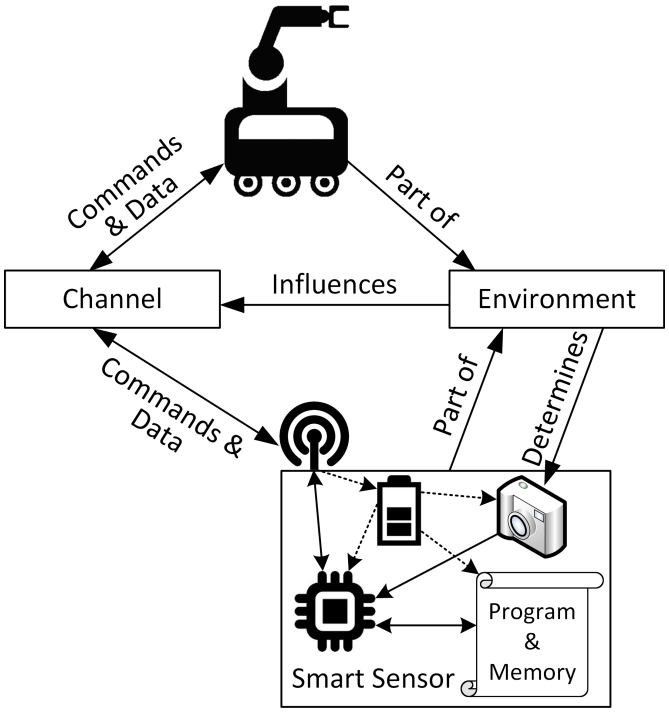

Fig. 5. Concept of the Evaluation Design.

The second challenge is the estimation of values that change during the skipping of time, such as the remaining energy in the battery. To estimate the values at the end of the time jump, detailed information about the process is required. As the time between two activations can be arbitrary, the error is unbounded. To mitigate that, a maximum time skip is defined. When linearizing the behavior of these transient values in the last instant, we can estimate the new value after the skip is completed. Depending on the maximum skip size, the final estimation can be very accurate.

4 Evaluation

The evaluation of the developed system is performed using the simulation of a smart sensor that charges the internal battery and communicates its information using NFC technology. The Gazebo simulator provides the context of the simulation. That is the environment in which the sensor is placed. During the startup of Gazebo, the developed sensor plugin is loaded and starts the SystemC simulation. A robot is placed outside the communication range of the sensor. The evaluation plan is to move the robot such that the sensor can be charged and communication can occur. When this is done, the robot requests data from the sensor.

For this evaluation we modeled the communication channel as a separate world plugin that can calculate the noises and signal attenuation due to the environment. This also allows us to change the channel parameters by swapping the

plugin. This also allows the reuse of the system for other communication technologies.

The communication between the two simulation environments is performed with strings encased in XML-tags. The received information is then stored in a data structure that groups the data blocks by the XML tag and stores the blocks in the order they arrive.

Different stimuli for the SystemC simulation can be combined, processed and sent by the developed plugin. Figure 5 shows the concept for this evaluation. The robot wants to send commands and data to the smart sensor via some channel (in this case NFC). To do so, it approaches the sensor, thereby changing the environment. This change influences the channel parameters. When the antennas are in close proximity to each other, data and energy can be transferred. The channel can, based on the parameters, change the data that is sent (e.g.: introducing bit errors). This message is then transmitted to the developed plugin. Furthermore, the plugin receives status information from the simulation, and parameters from the environment that the sensor can measure. This information is relayed to the SystemC simulation. The modified testbench processes the data to be used in the simulation. The simulation returns the results to the plugin. This plugin can manipulate the appearance of the sensor in the world (e.g.: switching on an LED), and transmitting the return messages to the channel. The channel again modifies the message according to the parameters and forwards it to the robot.

As the global simulation is done with a robotic simulator, a new test case can be implemented by changing the start position of the robot or introducing some randomness in the movement of the arm with the antenna attached to it. The rest of the simulation does not need to be altered in order to get new results. This simulation approach furthermore allows the testing of the interaction between the newly developed system and an existing (robotic) system. The evaluation of the correctness of the new system can also be done directly in the simulation as the robot expects certain answers.

As mentioned before, Figure 4 shows the compression of the idle time of the sensor showing the results of requirement R.9. Here the “advanceTime” trace is used to restore the time synchronicity of the two simulations after the simulation is finished. Whenever this trace peaks the simulation was stopped prematurely. The height of the peak indicates the time that is skipped. In this trace the first 7.1 seconds are condensed to about 0.6 ms. This means a reduction of memory and time usage of approximately 12000:1. Also the simulation can be stopped prematurely if the needed operations are finished before the time step is passed. This can be seen with the third drop of the “advanceTime” trace. This part is stored in 16.5 ms but refers to 0.15 s. This is a reduction of approximately 10:1.

Figure 6 shows the results of one simulation. Here the robot approaches the sensor until second 17. The antenna is activated from second 6.1 to 9. Here two messages are received (recData) but no return is generated, indicating that the channel has introduced errors in the sent message. This claim is substantiated by the fact that the received energy (EnergyIntake) is low and fluctuates. The fluctuation comes from the movement of the arm during the approach to the

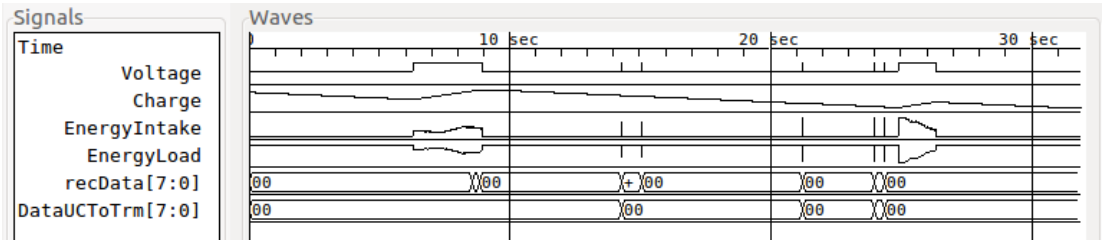

Fig. 6. Results of a completed simulation.

sensor.
The message at second 14.2 is being answered (DataUCToTrm). This means that the antennas are in a range where the communication can be successful. But the message at second 15.05 is not answered.
At second 21 the received energy is larger than before, indicating that the robot is close enough to communicate efficiently. In second 24, two messages are exchanged.
After second 25 the robot moves away. To show this the antenna is activated. During the retreat the received energy decreases.

The energy usage (EnergyLoad) of the sensor shows the combined energy budged of the sensor. As the energy gathered by the NFC antenna is much larger than the energy required for the calculations, it mirrors the energy intake. This also shows that the charging of smart sensors using NFC can be effective. The usage of the sensor itself can be seen in the trace of the remaining charge inside the capacitor (Charge). During the phases where the antenna is active, the charge rises, indicating that the capacitor is being loaded. In the mean time, the charge is slowly depleted as the sensor performs its operations with the energy stored in the capacitor.

Furthermore, the voltage available to the sensor (Voltage) is shown in this figure. Because the module used to charge the capacitor supplies the system with the maximum voltage allowed for this capacitor, the voltage rises rapidly to this value. In times where no energy is harvested the voltage is calculated using the charge of the capacitor, the energy currently used, and the serial resistance of the capacitor.

5 Future Work

A possible expansion of this system is another world plugin comparing the measurements of the sensor to the ground truth evaluated by the environment. This can be done by combining the information of the environment, the measured data from SystemC, as well as some of the messages received by the robot.
One drawback from forking the SystemC simulation from the Gazebo simulator is that the two processes are running on the same computer. If a simulation is created that encompasses multiple entities that are simulated using SystemC the simulations may need to share the same processor core, further slowing the

simulation. A solution to this would be to spread the SystemC simulation over a network and performing the communication using network sockets.

6 Conclusions

We presented an approach to connect SystemC simulations to the Gazebo simulator in order to automatically generate stimuli. This paper shows the difficulties that arise when connecting simulators that are designed to operate using different time steps. We showed a mechanism that can connect the simulations, proposed a mechanism that allows the interaction of the simulations, and formed requirements that need to be implemented on both sides to overcome the hurdles that we were presented by the simulations.
There are some core requirements that need to be changed we want to emphasize again. These include:

- *Synchronization* between the simulators is of utmost importance. SystemC operates usually more detailed and therefore needs longer to simulate one step. Gazebo must be halted while SystemC is running, otherwise the communication between the simulations can have unbounded delay.
- *Reduction of memory and time consumption* is important on all computers. Using our time reduction mechanisms, the realtime-factor of gazebo was optimized by a factor of 10^3 and the memory footprint reduced by up to 10^2.

This approach was first described by Pieber et al. [15]. In this publication, we extend the detail of the developed Gazebo plugin. Furthermore, we redefined some of the requirements needed for the SystemC adaptions to emphasize their purpose. Additionally, the evaluation describes how the systems interact and gives a detailed example of a complete simulation and what the created traces can look like.

Acknowledgements

This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 692480. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Netherlands, Spain, Austria, Belgium, Slovakia.
IoSense is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2016 and May 2019. More information https://iktderzukunft.at/en/

References

1. Accelera: SystemC. http://accellera.org/downloads/standards/systemc (2000), last accessed on Jan 17, 2017

2. Bombana, M., Bruschi, F.: SystemC-VHDL co-simulation and synthesis in the HW domain. In: 2003 Design, Automation and Test in Europe Conference and Exhibition. IEEE Comput. Soc (2003)
3. Bouchhima, F., Briere, M., Nicolescu, G., Abid, M., Aboulhamid, E.: A SystemC/simulink co-simulation framework for continuous/discrete-events simulation. In: 2006 IEEE International Behavioral Modeling and Simulation Workshop. Institute of Electrical and Electronics Engineers (IEEE) (sep 2006)
4. Huang, K., Bacivarov, I., Hugelshofer, F., Thiele, L.: Scalably distributed SystemC simulation for embedded applications. In: 2008 International Symposium on Industrial Embedded Systems. Institute of Electrical and Electronics Engineers (IEEE) (jun 2008)
5. Kirchner, T., Bannow, N., Grimm, C.: Analogue Mixed Signal Simulation Using Spice and SystemC. In: Proceedings of the Conference on Design, Automation and Test in Europe. pp. 284–287. DATE '09, European Design and Automation Association, 3001 Leuven, Belgium, Belgium (2009)
6. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566). Institute of Electrical and Electronics Engineers (IEEE) (2004)
7. Lee, W.S., Son, W.I., Oh, K.S., Yu, J.W.: Contactless energy transfer systems using antiparallel resonant loops. IEEE Transactions on Industrial Electronics 60(1), 350–359 (jan 2013)
8. Mathworks: Get Started with Gazebo and a Simulated TurtleBot. https://de.mathworks.com/help/robotics/examples/get-started-with-gazebo-and-a-simulated-turtlebot.html (2016), last accessed on Jan 03, 2017
9. Metta, G., Fitzpatrick, P., Natale, L.: Yarp: Yet another robot platform. International Journal of Advanced Robotic Systems 3(1), 8 (2006), https://doi.org/10.5772/5761
10. Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., von Stryk, O.: Comprehensive Simulation of Quadrotor UAVs Using ROS and Gazebo. In: Noda, I., Ando, N., Brugali, D., Kuffner, J.J. (eds.) Simulation, Modeling, and Programming for Autonomous Robots. pp. 400–411. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
11. Mueller-Gritschneder, D., Lu, K., Wallander, E., Greim, M., Schlichtmann, U.: A virtual prototyping platform for real-time systems with a case study for a two-wheeled robot. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013. EDAA (2013)
12. Nurseitov, N., Paulson, M., Reynolds, R., Izurieta, C.: Comparison of json and xml data interchange formats: A case study. Caine 2009, 157–162 (2009)
13. Open Source Robotics Foundation: Gazebo simulator. http://www.gazebosim.org (2004), last accessed on Jan 03, 2017
14. Panda, P.R.: SystemC - A modelling platform supporting multiple design abstractions. In: Proceedings of the 14th international symposium on Systems synthesis - ISSS. Association for Computing Machinery (ACM) (2001)
15. Pieber, T.W., Ulz, T., Steger, C.: SystemC Test Case Generation with the Gazebo Simulator. In: Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH,. pp. 65–72. INSTICC, SciTePress (2017)
16. Possadas, H., Adamez, J.A., Villar, E., Blasco, F., Escuder, F.: RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model. Design Automation for Embedded Systems (2005)

17. Strommer, E., Jurvansuu, M., Tuikka, T., Ylisaukko-oja, A., Rapakko, H., Vesterinen, J.: NFC-enabled wireless charging. In: 2012 4th International Workshop on Near Field Communication. Institute of Electrical and Electronics Engineers (IEEE) (mar 2012)
18. Sumaray, A., Makki, S.K.: A comparison of data serialization formats for optimal efficiency on a mobile platform. In: Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication. pp. 48:1–48:6. ICUIMC '12, ACM, New York, NY, USA (2012)
19. Willow Garage and Stanford Artificial Intelligence Laboratory: Robot Operating System. http://www.ros.org/ (2007), last accessed on Feb 15, 2018
20. Wireless Power Consortium, et al.: System description wireless power transfer. Volume I: Low Power, Part 1 (2010)
21. Zamora, I., Lopez, N.G., Vilches, V.M., Cordero, A.H.: Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo. arXiv preprint arXiv:1608.05742 (2016)

Chapter 8. Model-based Design of Secured Power Aware Smart Sensors

Thomas Wolfgang Pieber, Thomas Ulz, and Christian Steger

Abstract When designing new smart sensor platforms, the system should be well adapted to the intended use case. In most cases this means that the sensor will be implemented as a part of a larger system - be it as a part of a sensor network or a component of a machine. In both cases the sensor should have a long lifetime, use the available resources with care, handle the data securely, and prevent the system from getting damaged by misusing the sensor knowingly or unknowingly. To test all of these properties, models of the sensor (and its component parts) can be created and used in simulations that represent the environment and the possible uses of the sensor in it. This chapter describes the possibility of creating a new power aware and secured smart sensor using a model-based design approach.

Key words: Model Based Design, Simulation, SystemC, Gazebo, Prototypes, Smart Sensors, Secured IoT

1 Introduction

Sensors can be found everywhere in our society. They measure the environment, gather useful data to navigate robots through the environment, and enable machines to sense the environment they want to interact with and record the changes they cause. To get better at these versatile tasks, new sensors are developed steadily.

Thomas Wolfgang Pieber
Institute for Technical Informatics, Inffeldgasse 16/I 8010 Graz, e-mail: thomas.pieber@tugraz.at

Thomas Ulz
Institute for Technical Informatics, Inffeldgasse 16/I 8010 Graz, e-mail: thomas.ulz@tugraz.at

Christian Steger
Institute for Technical Informatics, Inffeldgasse 16/I 8010 Graz, e-mail: steger@tugraz.at

Original work published in Sensor System Simulations, pages 241-256. Springer International Publishing, 2019.

To develop new sensors efficiently, simulations of the intended use case are created and model simulations of the new sensor are placed in the simulated environment. Research prototypes can then be used to verify the simulation results. If the simulation results do not resemble the real measurements, the prototypes can be used to search for errors in the simulation and gain new insights in the processes of the use case.
This model-based design approach of sensors can be used to efficiently create optimized sensors for any use case.

The act of sensing, as well as the processing, storing, securing, and transmitting of the gathered data requires energy in form of electric power. This energy can be provided by various methods. For many sensors this energy is provided by electro-chemical batteries or is gathered from the environment by energy harvesting methods. These forms of energy provisioning have in common that the sensor needs to manage its available power carefully as either the energy is limited or only a limited amount of power can be extracted from the environment. When designing a sensor that should be operated with one of these methods, it is necessary to know the characteristics of the component parts of the sensor and the possibilities of the energy provisioning system with high accuracy. These parameters can influence the design of the sensor massively. That means that the components of the sensor need to match the requirements for the intended use case and the energy provisioning needs to be capable of providing sufficient energy to operate the sensor for the intended lifetime and operations. To work efficiently the sensor has to be aware of its power usage and the current capabilities of the energy provisioning system.

The data that is generated by the sensors is mostly seen as unharmful to the process they monitor. This can be seen as the most common approach to data security of sensor data is "*What can be measured can be seen by anyone*". This notion of data security is now seen as a fallacy as the *STUXNET* [9] and *HAVEX* [4] attacks have been noted. Furthermore, as the *Mirai botnet* [2] shows, unsecured sensors also pose a threat to other targets. To circumvent these threats, measures to secure the system and the generated data need to be taken.
To prevent the new sensor to be exploited in such way, the security of the system and data is of utmost importance, and thus, must not be neglected during the design of the new sensor system.

The paragraphs above describe a sensor that can not only generate data about the environment, but also about itself and take informed decisions. It can furthermore modify the data and perform security relevant operations on it. Beyond that, it can receive and send such data to form a network. It also knows its current power status and can perform actions to prolong its lifetime or let the network know if the remaining energy gets too low.
Such a sensor that is able to perform informed measures on itself, the data, or the network can be called a smart sensor.
This chapter follows the design process shown in Figure 1.

- A reference system is created that shows all processes we want to model.
- The reference system is measured to understand the processes.
- A simulation of the system is created to abstract the system.

Chapter 8. Model-based Design of Secured Power Aware Smart Sensors

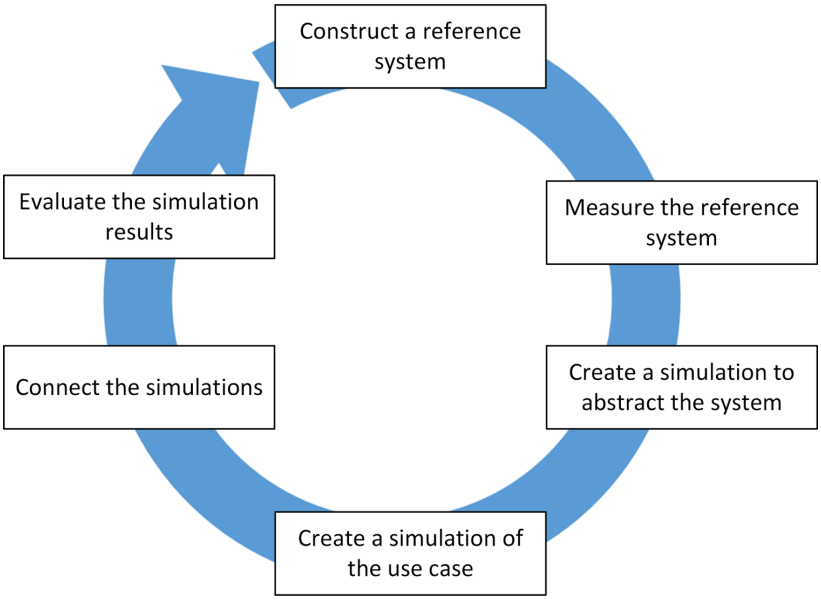

Fig. 1 Concept of a model based design process for a secured power aware smart sensor.

- The use case is modelled and simulated.
- The simulation of the new system is connected to the simulation of the use case.
- The results of the simulations are evaluated. If the results are not as expected the process is restarted.

In this chapter we use this design approach to create a sensor that is read and charged by using NFC technology. To do so autonomous robots in an *Industry 4.0* inspired setting will communicate to the sensors and connect them to a network.

2 Related Publications

Simulation Background

The use of simulation to predict sensor functionality and the methods to get the parameters of the sensor components have been described in many publications.

In this publication we want to use the robotic simulation tool Gazebo [14] as a core system. The connection of a Matlab environment to Gazebo has been published on the official website [12]. This tutorial shows how to connect Matlab to Gazebo via the ROS (Robot Operating System) interface to send and receive data from the Gazebo simulation.
Further publications connect the Gazebo simulator with different software for machine learning [26]. Also here the software is connecting to the operating system controlling the robot and influences the path the robot takes to reach a goal.

SystemC [1] is a hardware description language based on C++. It is capable of describing the hardware on different levels of abstraction. This is especially useful as a detailed description of all components results in a very slow, albeit accurate, simulation. This is furthermore useful if developing a simulation from scratch as the components can be described in an abstract way and be defined in later steps. The publication of Panda shows design processes to create SystemC simulations [15].

The connection of SystemC to different simulation tools has been described in many publications [3, 7, 11, 13, 18]. In these works the SystemC simulation is primarily used as a kernel to provide the functionality of the quasi parallel execution.
The connection of SystemC to Matlab was designed by Bouchhima et al. [3] As Matlab works with discrete events, they created this cosimulation to additionally simulate continuous events.
To speed up the SystemC simulations, Huang et al. [7] describe a possibility to spread the simulation across a network of computers.
SystemC can also be connected to a simulation tool designed for integrated circuit simulation[11]. Martin et al. connected it to a SPICE (Simulation Program with Integrated Circuit Emphasis) simulator as well as to an VHDL (Very high speed integrated circuit Hardware Description Language) simulation.
Mueller-Gritschneder et al. [13] used SystemC to compute physical processes that affect the robot while it is moving. This simulation therefore generates data in the SystemC simulation that is then used to model the robots behaviour. In contrast to that, our approach generates data in the environment and uses this as input for the SystemC simulation.

Pieber et al. [16] describe methods to connect a SystemC simulation to the Gazebo environment in one of their publications. This paper describes in detail how the connection between the simulations can be formed and what requirements need to be fulfilled to create a successful simulation. This publication furthermore announces possible advancements to increase the simulation speed if multiple SystemC simulations need to be run at the same time.

Power Awareness for Sensors

The idea that sensors are aware of their own power levels and power consumption has been discussed in many publications [5, 8, 10, 19, 20, 25]. Using this information, the sensors can make decisions about the routing of information inside the network, reduce the quality of the measurements to prolong the lifetime, or inform the network that the energy level is too low to actively take part in the task of the network.
Chen et al. [5] developed a small energy harvesting sensor node that is capable of providing its own energy. This sensor node relies on solar cells to charge a battery. If the charge in the battery is sufficient the sensor wakes and performs its operations until the voltage drops below a predefined value. Then the sensor switches to a ultra-low power sleep mode. In this scenario the sensors duty cycle is provided by the capabilities of the energy harvesting system.

To calculate a duty cycle for a energy harvesting sensor system the energy usage and harvesting capabilities must be known. Kansal et al. [8] formalized these calculations in their work. These configurations can be used to initialize such system. If the observed parameters at the position of the sensor differ from the predicted ones, the team of researchers propose a dynamically changing duty cycle. Using this dynamically changing system the researchers noted a significantly improved performance of their research system.
Additionally Rahimi et al. [19] explore the possibility to expand the sensors lifetime using energy harvesting methods. In their research they try to exploit autonomously moving nodes. These nodes search in a so called "Energy Cell" for the optimal spot to harvest solar energy. Afterwards the moving nodes provide this energy to the sensor nodes in their assigned cell.
Another possible energy source that can be used by the sensor nodes is RF energy from broadcasting stations. Sogorb et al. [20] use sensor nodes with two antennas to research this possibility. In their research prototype one antenna is used to harvest the energy from broadcasting stations, the other one to transmit the gathered data to a base station.
Another strategy that uses RF energy harvesting has been explored by Lee et al. [10]. This research team focused on the possibility to harvest energy using antiparallel resonant loops. The researchers explore RFID's (Radio Frequency IDentification) capability to transmit energy. They stated that antiparallel loops can improve the efficiency of energy transmission to 87% from about 50%.
The researchers around Yan [25] explored methods to construct more efficient sensor networks. This is done by making the sensors aware of their own energy state. In this research Yan et al. implemented two levels of energy saving. "*Node-level energy saving*" adaptively regulates the transmission power of the sensor node. "*Network-level energy saving*" adaptively reconfigures the sensor networks configuration.

Security for Sensors

Security for sensor devices has been researched in the literature [6, 17, 22, 23, 24]. Most researchers however see security as a side topic to be added later to an existing sensor hardware.

The work by Ulz et al. [24] describes multiple methods how sensor data can be secured. The researchers pointed towards problems that arise if sensor data is encrypted and how these problems can be mitigated. In their publication the researchers use authenticated encryption (AE) to securely transmit data between a sensor of a control system and the controller. The researcher state that using only this technique a DoS attack is easier to perform. To counteract this problem forward error correction techniques have been proposed. Furthermore, the researchers state that a security controller can be used to perform these operations as this processing unit is specialized to perform the task of encrypting more efficiently than a regular microcontroller.
In addition to securing the data of a sensor against adversaries Ulz et al. [22] pro-

posed methods to secure a sensor network against misuse and misconfigured sensors using a security controller. This paper presents the idea of a two-layer attestation system that first checks the validity of a sensors firmware, and in a second step validates the version of this firmware. Another publication by this researchers describes an update mechanism for such sensor system [23]. Here the researchers state that the interface to update the sensor needs to be separated of the main communication interface.
To prevent the misuse of single sensors, authentication to see and modify data is necessary. Pieber et al. [17] describe a method to use a password-based authentication method that is lightweight enough to be run on a single sensor.
Haase et al. [6] propose a system to (re-)configure a sensor system via NFC . They furthermore use cryptography to authenticate the reconfiguration device.

3 Obtaining Data for the Models

To get a decent simulation of any system, the processes and components of the system need to be understood. A smart sensor consists at least of the following parts:

- The sensor frontend: The part of the sensor that does the conversion from an external stimulus to digital information.
- The controller.
- The memory.
- A communication interface.
- The energy provisioning system.
- (Optional) A security controller to perform security relevant operations.

For these components data sheets are available that specify the peak current at an optimal voltage source. This is sufficient information to build the sensor, as overestimation of the components energy usage leads to a more powerful energy provisioning system. To get a better estimation of the energy demands of the system, and therefore be able to design a mote optimal energy provisioning system, measurements have to be performed. This requires the development of a research prototype.

To create an efficient sensor, a model of the environment and the intended use case is useful. Such simulation need not be very detailed as this allows the simulation to be used in various ways and thus create a multitude of different stimuli for the sensor. The simulation of the environment needs at least these components:

- A representation of the new sensor.
- Objects influencing the sensor or the communication with the sensor.
- Communication partners for the smart sensor to test the used communication protocols.
- Things for the sensor to measure.

Chapter 8. Model-based Design of Secured Power Aware Smart Sensors

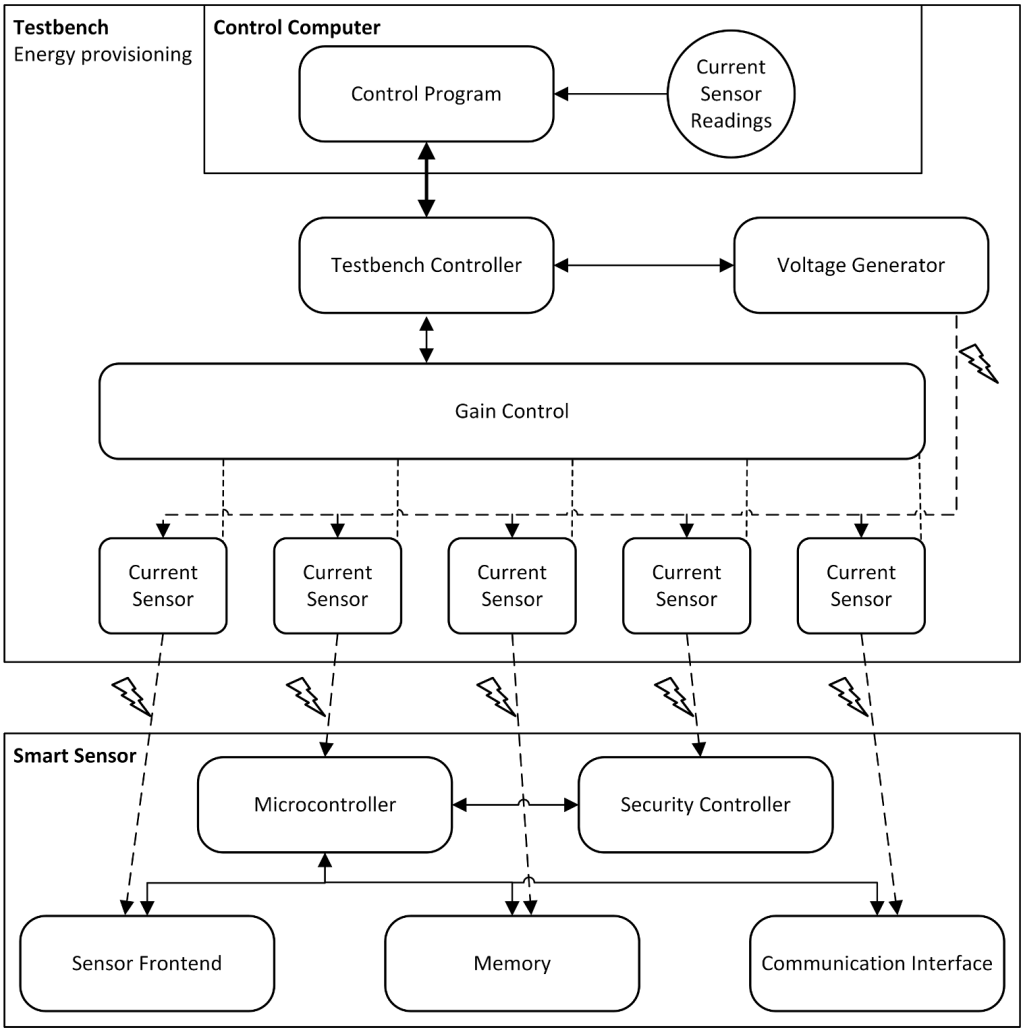

Fig. 2 Concept of the measurement system for a smart sensor.

3.1 Measuring Data for the New Sensor

The prototype that is used to measure the sensor components should be able to vary the input voltages to emulate the energy provisioning system. This concept is shown in Figure 2. Here a control program gets the values of the current consumption of the components and calculates how the voltage of the energy provisioning system reacts to this current flow. The updated settings are transmitted to the testbench where the voltage is generated. The smart sensor can now be operated with this updated voltage and the current flow changes. To have an optimal measurement of the drawn current a gain control unit can be used to set the sensor gains of the current measurement units. The measured values are given to the control program to be stored and to calculate the new settings for the testbench. The data gained from the measurements are used to construct models of the sensor components.

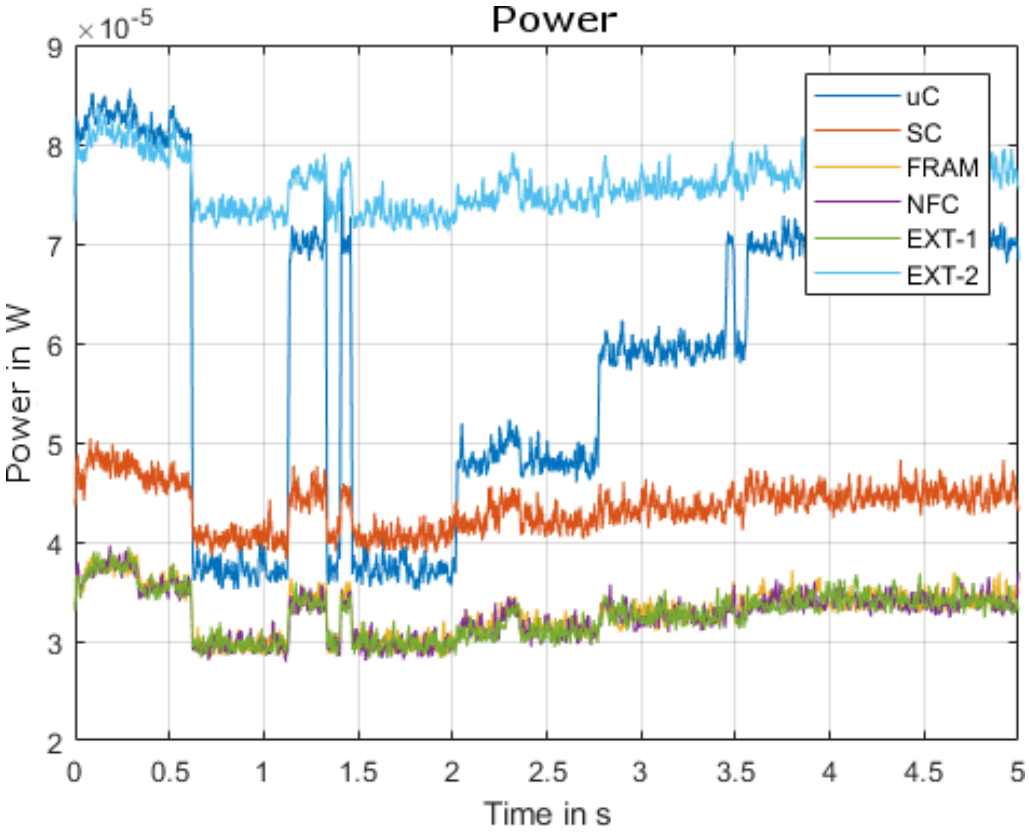

Fig. 3 Example of a measurement of sensor components.

The data that is generated can then be used to create models of the sensor components energy usage at different voltages. This data can furthermore be used to specify the requirements for the energy provisioning system of the final sensor.

3.1.1 An Example

To create an energy efficient sensor, all components should be optimized for low energy usage. Furthermore, components that are not needed still consume energy. To counteract that, load switches can be used to cut the components off the power supply. In Figure 3 the same sensor device is connected to the channels *EXT-1* and *EXT-2*. *EXT-1* is cut from the power supply with a load switch. This sensor uses 33 µW. If it is switched on the sensor consumes 75 µW.

Such simple method can reduce the energy consumption drastically and therefore prolong the sensors lifetime.
The only component that cannot be cut from the power supply is the microcontroller. There the energy consumption can be reduced by switching into a low-power state. Figure 3 shows a measurement of a smart sensor. Here the sensor connected to *EXT-*

Chapter 8. Model-based Design of Secured Power Aware Smart Sensors

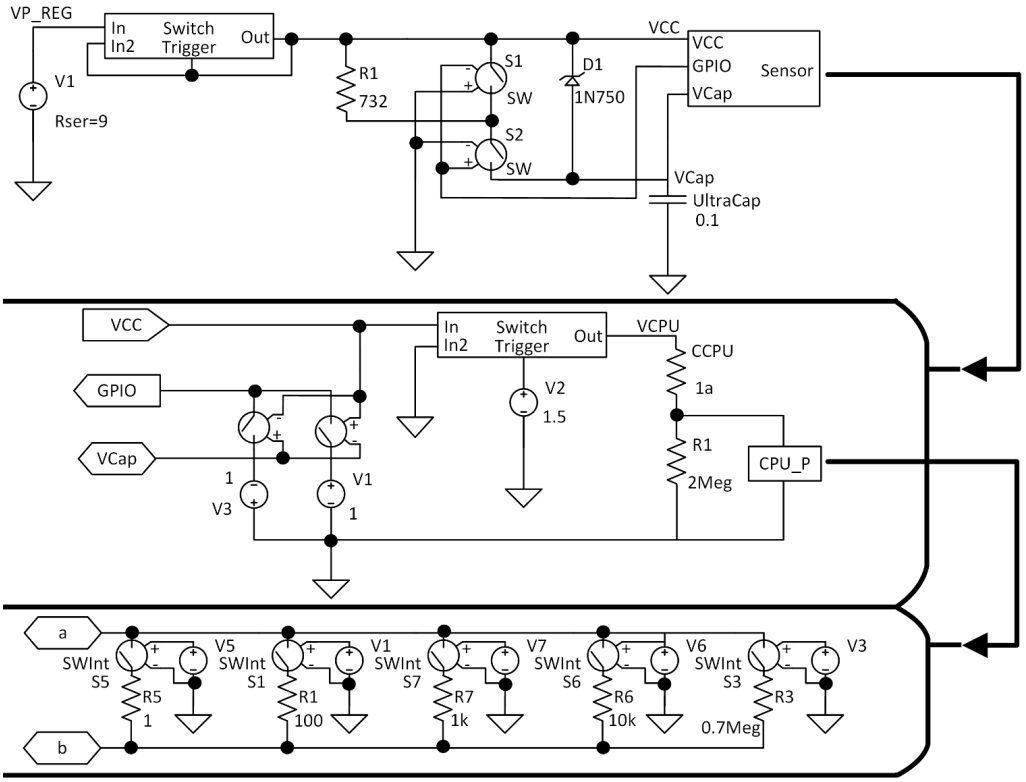

Fig. 4 Electrical model of a smart sensor.

2 is continuously active. LEDs, connected to the microcontroller, switch according to the measured value.
Additional memory may be useful for a sensor system. Here configurations can be stored and measurements can be buffered until they are transmitted. For the FRAM memory module in this design the energy consumption in standby is around 35 µW. To communicate with other devices and to harvest energy, this example design features an NFC interface. This interface consumes about 33 µW in idle mode.
Stated as an optional component is a security controller. The energy consumption of such element is around 45 µW in an idle phase. Also here a load switch can reduce the energy consumption and prolong the sensors lifetime.

Using this approach, the energy provisioning system is replaced by the testbench. That implies that the energy provisioning needs either to be measured using a different method, or an existing simulation can be used to get to the needed data. There are numerous simulations of energy harvesting methods and batteries available that can be used for this purpose.

To simplify the simulation of the electric system of the sensor, an intermediate simulation tool can be used. Figure 4 shows how the energy consuming parts of the sensor can be modelled in LTSpice.

The top part represents the energy harvesting and energy storing of the sensor system. Here, the input voltage (*V1*) represents the capabilities of the energy harvest-

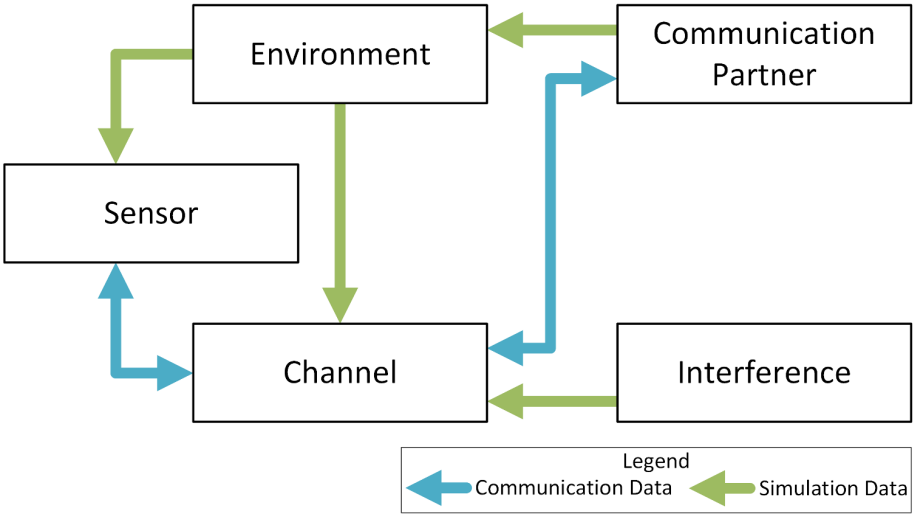

Fig. 5 Example of an environment of a sensor.

ing device. If this voltage is higher than the sensor's supply voltage the voltage from the harvesting device is switched to the supply voltage. The sensor is then driven by the harvested energy. When the sensor system (*Sensor*) detects that the supply voltage is larger than the voltage at the energy storage (*UltraCap*) the switches *S1* and *S2* are flipped and the energy storage is charged. If the harvested voltage gets below the voltage from the capacitor, the switches are flipped again and the sensor is not charged any longer. If the harvested voltage drops below the voltage that can be supported by the energy storage the sensor is again driven by the stored energy.
The middle section of Figure 4 shows the mechanism that flips the switches in the top section (*S1, S4, V1*, and *V3*). Additionally, the mechanism that completely switches off the sensor if the voltage gets too low (*Switch* and connected *V2*) is implemented in this section. The operational part of the smart sensor (*CPU_C*) is then driven with that voltage.
In the bottom section of this figure various electrical networks (in this case resistors) can be switched on and off. These networks are tuned to represent the different components and power states of the sensors components.

This reduced electrical model of the sensor can be used to model the energy consumption of the sensor components in the final simulation.

3.2 Gathering Data for the Environment Simulation

The environment simulation will be acting as the de-facto testbench of the sensor simulation. Therefore, it should allow a multitude of different scenarios. The environment simulation can be created with a low complexity and still produce a good quality result.

Figure 5 shows the most crucial parts of the environment simulation. In this figure, the blue arrows represent the data path for the communication between the *Sensor* and its *Communication Partner*. The green arrows represent the data that is generated in the environment simulation. Here the *Communication Partner* is a part of the *Environment* and can manipulate some variables of it. Most notably the *Communication Partner* can manipulate its own position. The change in the *Environment* influences the parameters of the *Channel*. Furthermore, this parameter change influences the data the *Sensor* can observe. The *Channel* receives information about the *Environment* and the *Interference* generated by other communications. Using this information the *Channel* modifies the data that is communicated between the *Sensor* and its *Communication Partner*.

The representation of the sensor needs to be placed somewhere in the world as reference point. All stimuli for the sensor are calculated in reference to that point.
To validate the communication interface of the sensor, a communication partner needs to be introduced to the virtual environment. For a better simulation of the communication, interference generators and obstacles to the transmission may be included.
The validation of the sensor interface can be performed by including the measured variable in the simulation. When simulating the capabilities of an energy harvesting device, also this stimulus needs to be generated in the simulation.

This virtual environment allows the testing of the sensors use case. The environment can be modified to test different use case scenarios, boundary conditions of the communication, and reaction of the sensor to faulty signals.

4 Creating the Simulations

When designing a new sensor, simulations play an essential role. Not only the simulation of the sensor system, but also a simulation of the intended use case is useful.

4.1 Designing the Sensor Model

Using the data gathered in Section 3.1 a model of a generic sensor can be created using SystemC. This models needs to represent all features of a smart sensor. These features are:

- **The sensor frontend:** Depending on the use case the smart sensor can be equipped with different frontends. Typical frontends can just be switched on or off thus it can be modelled as a static energy consumer.
- **The control unit:** The microcontroller can be represented as a timed state machine. Different power states that the microcontroller operates in can be defined (e.g.: calculating, memory access, idle, sleep).

- **The memory:** If the sensor includes an additional memory unit, a detailed description is needed. Typical memory modules have very few operational modes (data access, data write, sleep). These can be modelled very accurately without having an overly complex model.
- **The communication unit(s):** The communication is typically slow and energy intensive. The model for the communication needs to be as accurate as possible. The description of the hardware itself can again be abstracted as a timed state machine operating with different power levels (sending, receiving, sleep, energy harvesting).
- **The energy provisioning system:** Many sensors are either connected to a bigger machine or to the power grid. For those sensors we can assume the voltage to be constant as it should be possible to provide sufficient energy to the sensor. If the sensor is operated by a battery or using an energy harvesting system and a capacitor, this system needs to be modelled with high accuracy. Voltage drops caused by to high energy demand can cause the sensor to stop operating. To counteract this, the sensor can use measures to decrease the energy consumption by temporarily disabling some functionality. In addition to these radical changes to the energy consumption, the non-linear behaviour of capacitors and batteries is important to the accurate simulation of the voltage levels of the sensor.
- **A security coprocessor:** The modelling of a security coprocessor can be tricky as the detailed description of it is most of the times classified. Nevertheless, a model of it needs to be constructed, as this coprocessors typically consume much energy to conceal their behaviour. The security coprocessor, can be modelled as a static energy consumer when switched on. This overestimates the real consumption and leads to a more robust design.

After these components have been created, they need to be connected to each other. Figure 6 shows how these connections can be set. The red arrows represent the energy transfer between the components. In the case that NFC is used as means of communication, energy can be harvested from the RF field. This energy is stored in the sensor's capacitor or accumulator. The antenna system also requires some energy if it is not supplied by an external field. All other components have to be connected to the energy supply. The blue arrows in this figure represent the data connections. All components are connected to the microcontroller. Either, as represented here directly, or via a bus system.

When creating this simulation a tradeoff between accuracy of the model and the simulation speed needs to be made. The more complex the model of the smart sensor becomes, the slower the simulation will be. In addition to reducing the complexity of some of the sensor components models, optimizations for longer simulation periods where the sensor is idle need to be made. These optimizations require the knowledge of the sensors non-linearities, the stimuli that wake the sensor from sleep, and the actions the sensor takes when stimulated.
Such optimizations can stop the execution of the full simulation and switch to a reduced version, or just estimate how the transient variables would behave during these periods. When estimating the variables, non-linearities in the sensor system need to be accounted for.

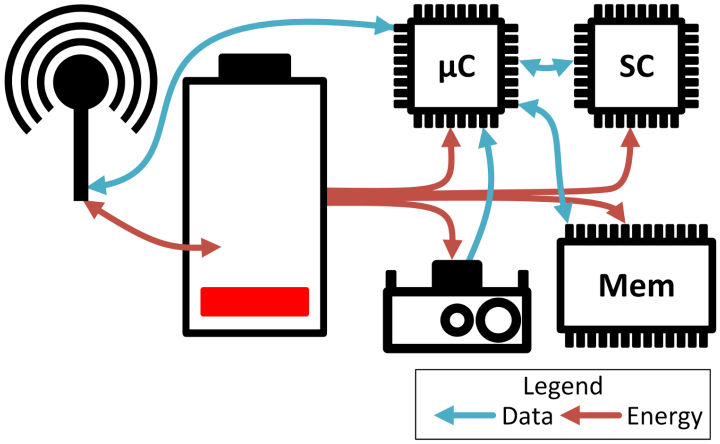

Fig. 6 The overall structure of a smart sensor.

4.2 Designing the Environment

To test if the new sensor will be able to perform all tasks of the intended use case, an environment needs to be created that can influence the simulation for the sensor. The environment can be simulated using the "Gazebo" simulator. To describe the different actions an entity can take, plugins need to be created. This simulation can then be used as a testbench for the sensor simulation. Using the data from Section 3.2 a model of the environment can be created. Its vital components are:

- **The sensor:** A representation of the sensor in the environment. This component implements the interface to the sensor simulation. All interaction of the environment with the sensor are relative to this representation. Additionally, all parameters the simulation calculates for the sensor to measure are calculated with this reference model.
- **Something to measure:** A simulation tool representing 3D objects primarily computes the relative motion and size between objects. Other parameters such as communication signals, air humidity, temperature, or light might not be calculated. Depending on the use case of the sensor these variables need to be calculated and passed to the sensor model.
- **A communication partner:** To test the communication unit of the sensor a counterpart needs to be implemented in the environment. This can be performed by another instance of the new sensor or an already existing device. To be able to test different scenarios it can be useful to allow the communication partner to be moved.
- **Interference and obstacles:** Any communication is influenced by the environment it is performed in. In the case of a wireless communication interference from other machinery, nearby communication, signal attenuation, and the scattering and reflecting of the signal produce communication errors. To see how

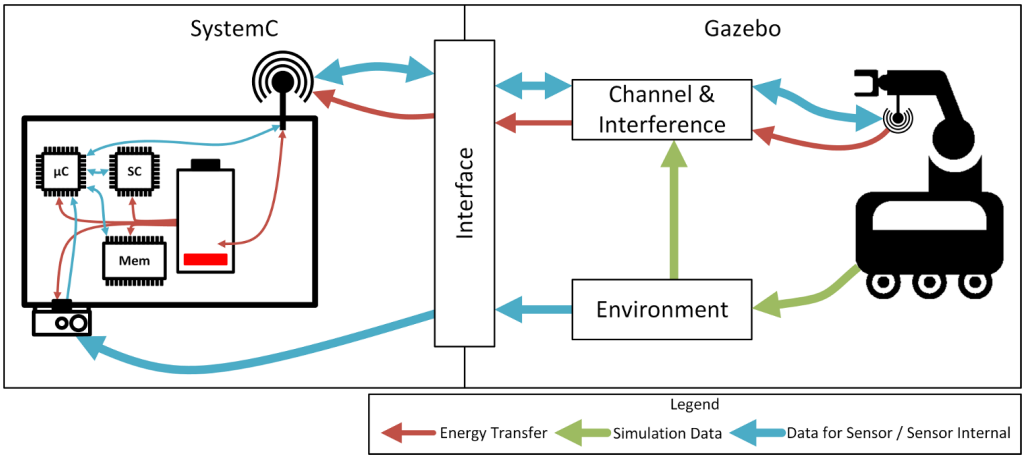

Fig. 7 The connection of the sensor and environment simulation.

the sensor reacts to faulty messages an implementation of the communication channel, including some interferences, is needed.

These components are then placed in a 3D virtual environment and logically connected.
The sensor receives information about the communication from the implementation of the channel and information about the measured variable directly from the environment. This data is then forwarded to the sensor simulation. The data that is produced by the sensor simulation is filtered and forwarded to the intended destination. This destination can be an actuator or the communication channel.
The communication partner sends the information that is intended for the sensor to the communication channel and receives the information from the sensor via this channel. If the communication partner is movable the information about the changing position is sent to the environment.
The channel receives data from all communication partners and noise sources. Additionally, information about the 3D environment is gathered. Using this information the received data is modified (bit errors are introduced and the transmitted energy is calculated) and sent to the receiver.
The environment gathers data about the location of all objects. The information about the measured variable is then calculated and sent to the sensor.

4.3 Inserting the Sensor in the Environment

The simulation of the sensor and the simulation of the environment are very different simulation types. On the one hand a simulation of a hardware and the processes that occur in the hardware during the operation. On the other hand a simulation of some physical processes, movements, and communication. The major difference in these simulations are the time steps in which the simulation operates. The simulation of

the hardware can occur at a very low level of abstraction and thus the time steps are small. SystemC for example supports time steps as small as 1 fs. The movement of robotic appliances is simulated in larger time steps. Gazebo, a robotic simulation tool, operates in time steps of 1 ms. This difference in simulation speed requires a complex *Interface* between the simulations.

Figure 7 shows the overall structure of the connection between the Gazebo simulation of the environment and the SystemC simulation of the sensor. The red arrows in this figure represent the flow of energy in the simulations. The blue and green arrows show how data is transmitted between the modules. Here, the blue arrows represent information that is transferred to or from the sensor, the green ones information that is required by the environment.

To create a connection between the 3D representation of the sensor in Gazebo and the simulation of the sensor in SystemC, a plugin is connected to the Gazebo model. This plugin forms the *Interface* between the simulations. To perform the task of connecting the two simulations, all the gathered data is packed and transferred to the SystemC process and the returning data is distributed in the Gazebo simulation. This plugin furthermore handles the synchronization between the simulations. This is done by adding additional information about the time step and a signal that the SystemC simulation should perform the calculations for this step.
In the SystemC simulation the original testbench is replaced by the counterpart of the *Interface*. It receives the information and forwards it to the destination. The destination for the measured variable is the model of the sensor frontend. The data that is sent via the communication channel is forwarded to the communication interface. The additional information about the time step is kept at the testbench. With the information about the received values the testbench can calculate whether the full simulation needs to be run or if the changing parameters can be estimated. During the calculations of the simulation messages may be created that are then sent to the Gazebo simulator. When all calculations for this time step are finished the testbench sends a signal that the Gazebo simulation can perform another step.

Beyond the Immediate Neighbourhood

Using the data of the sensor system and the immediate interaction with the environment, more simulations can be created. Theses simulations can model the behaviour of a network of such sensors. The sensors in this network can be placed in a larger scale environment with moving communication partners transporting data and delivering energy. The approach of Ulz et al. [21] to form a network using mobile communication partners can here be used to also distribute the required energy to the sensor nodes.

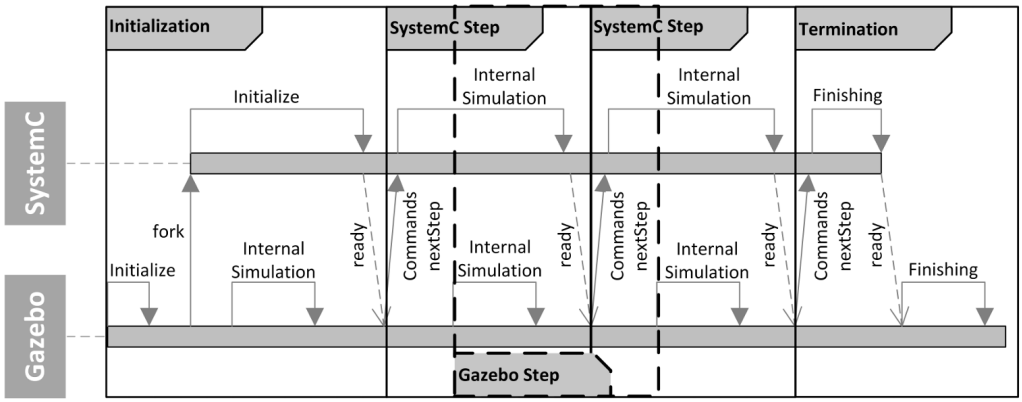

Fig. 8 Sequence for the Gazebo-SystemC cosimulation.

5 The Simulation in Detail

In the Gazebo simulator all operations are performed in plugins that are connected to entities in the simulation. These plugins are executed sequentially such that the plugin that is being executed can already work with the results of the plugins executed before.
To synchronize the two simulations Gazebo needs to be halted until SystemC has finished and SystemC needs to wait for Gazebo to provide the parameters for the new time step.

The simulation sequence is shown in Figure 8. This figure shows the initialization phase, simulation steps, and the termination of the cosimulation.

Initialization

When the simulation is started all Gazebo plugins run an initialization routine. In these routines the communication structures between the plugins are created and initial values are loaded. In the case of the environment description, all initial positions are gathered. The implementation of the channel loads the relative positions of the antennas and obstacles. The interface implementation forks a new process that will execute the SystemC simulation. It furthermore spawns a new thread that will be listening to the forked process. Finally, initial configuration parameters are sent via a communication channel.
The newly created process starts the SystemC simulation, loads the initial configuration from the provided communication channel and informs its parent process, the Gazebo simulation, about its status and waits for instructions.

Simulation Steps

The Gazebo plugins that control the communication partner check if inputs from a user or the robot's operating system are available. These commands are then translated to actions the communication partner will take. These actions include moving itself or sending some messages.
The plugin controlling the environment receives the informations about movements and calculates new input for the sensor.
If messages are sent, the channel plugin gathers the information about the relative distance and orientation between the communicating parties and obstacles. Using this information the messages are altered to include bit errors. Additionally the transmitted energy gets changed to account for the channel properties. The modified messages are then forwarded to their destination.
The interface implementation waits for the SystemC simulation to have finished its last simulation step. When this signal is received the gathered information is forwarded towards the intended destination. After that all information that is sent to the interface plugin is packed and sent via the communication channel to the other process. This information is appended by information regarding the time step and a signal that informs the SystemC simulation that all information is sent. The SystemC simulation can now execute the needed operations while the Gazebo simulator computes the information needed for the next simulation step. Parallel to the computation of the next step, the thread of the interface listens to the channel between the processes for information the SystemC simulation sends to the Gazebo environment.

The SystemC simulation receives all information sent to it by the Gazebo simulator. The testbench can now adapt the simulation parameters and start the simulation. While the simulation is being executed messages can be sent to the Gazebo simulator. These messages can originate from the communication interface module. If this sensor also includes an actuator or some visual status indicator these modules can also send information to the environment simulation.
When the simulation step is finished the testbench sends the signal that the simulation step is finished.

Termination

After the simulation has been performed the plugins receive a signal that the simulation is about to be ended.
The interface plugin relays this information and waits for the remaining messages from SystemC. When it receives a signal that the SystemC simulation is terminated the plugin cleans its data structures and terminates.

When the SystemC simulation receives the signal to terminate, a post-processing step for the gathered data is initiated. This post-processing is a part of the runtime optimizations and performs tasks to correctly display the gathered data. The data is

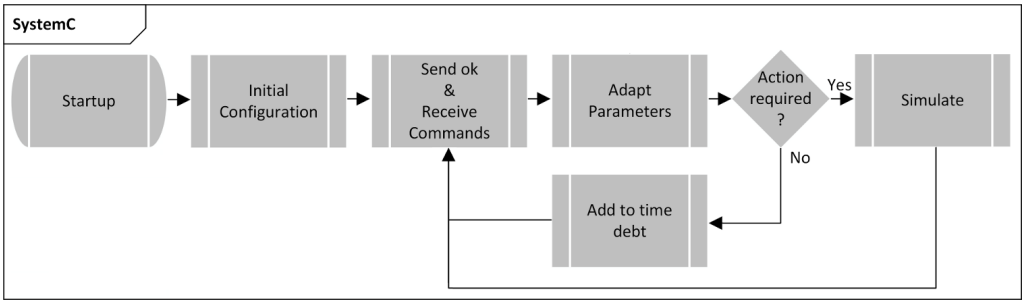

Fig. 9 The testbench of the SystemC simulation is adapted to skip the simulation if it is likely to yield results that can be estimated.

furthermore compressed by the post-processing step. Finally the data is saved and the Gazebo simulation is informed of the termination of the simulation.

5.1 Optimizations

As the SystemC simulation operates in smaller time steps, each step of the Gazebo simulation causes thousands of SystemC steps. This results in the SystemC simulation being much slower than the Gazebo one, and thus slowing down the whole process. To speed up the simulation, the accuracy of the results can be reduced at certain times. A possible method to reduce the accuracy is shown in Figure 9.

To take advantages of this possibility, the testbench needs to estimate the state after the simulation step. If the sensor is in idle mode and the input parameters do not change, the accuracy can be reduced by skipping the simulation and keeping a number that describes the amount of time that has been skipped since the last simulation step. If the estimator evaluates that a simulation is needed this *time debt* is used to calculate changing parameters. To account for non-linearities in the simulation that have not been included in the estimation process a simulation step is needed if the *time debt* exceeds a defined time interval.
As SystemC is not created to support this behaviour, the output files of the simulation need the post-processing to be correctly displayed.

6 Evaluating the Simulations

The first step in the evaluation of the simulation results is the verification that the components of the simulations show the same characteristics as the measured ones.

The results of the LTSpice simulation are compared in Figure 10. Here the left image shows the measured variables of a research prototype. In this scenario the sensor was charged and performed measurements every 60 s. The right hand side shows the same operation simulated using LTSpice. The direct comparison shows

Chapter 8. Model-based Design of Secured Power Aware Smart Sensors

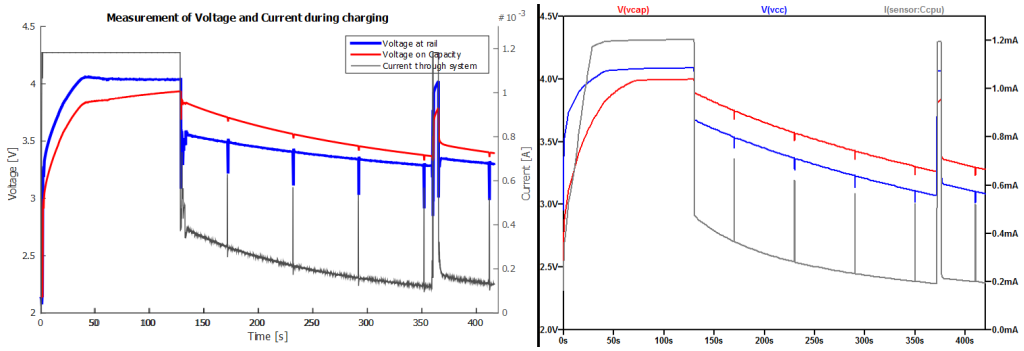

Fig. 10 Comparison between the measured values during charging and the electrical LTSpice simulation.

that the LTSpice simulation approximate the real system to a high degree.
There exist some possibilities to optimize the energy consumption of the sensor itself.

- The duty cycle can be adapted to reduce the number of measurements. The spikes in the sensors current indicate the measurement.
- Some of the sensor components can be cut off the energy supply. For example the sensor can take many measurements and keep the values in its internal storage. If this storage gets too full, all data is transferred to the external memory module. To do so, the module is then switched on and all data is transferred. This reduces the amount of time the memory module spends in its startup phase.
- The communication module only needs to be switched on if some communication is about to happen. The same principle applies to the security controller of the sensor.

Figure 11 shows a comparison between the simulation in LTSpice and the simulation performed in SystemC. The left hand side of this figure shows the results of another LTSpice simulation. The right hand side shows the same process in the SystemC simulation. Also here the capacitor has been charged and after that a simulated measurement has been performed every 60 seconds. The results show that the two simulation behave similarly. To optimize the energy consumption of the complete sensor system, some more options exist.

- The more valuable the measured variables are, the more often the sensor needs to be looked after. If the significance of the variable changes, the sample rate can be adapted to reduce the energy consumption of the sensor. Additionally, the sensor can change the interval in which the measured values are transmitted. This can be changed through configuration of the sensor.
- If the data is transmitted using mobile nodes, the moving partners can optimize their routes to gather the data while they are on the way to perform some tasks.

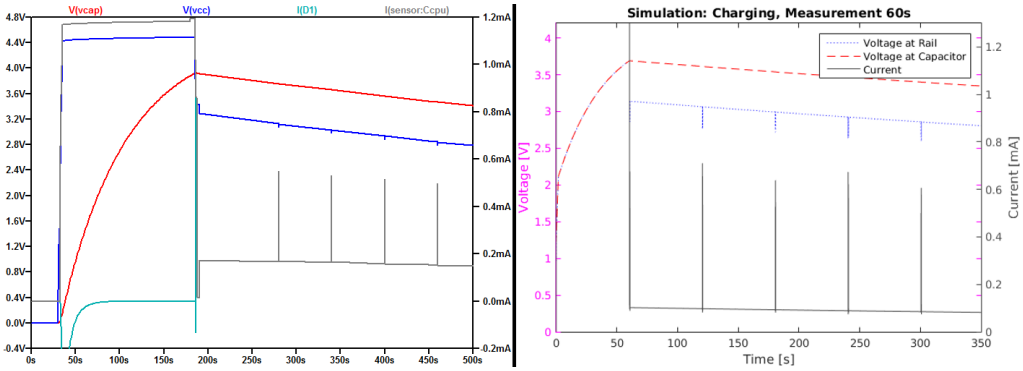

Fig. 11 Comparison between the measured values and the results of the SystemC simulation.

Examining the Effects of the Optimizations

The runtime optimization that can skip simulation steps introduces some problems. Figure 12 shows two traces to examine this effect. The top trace is taken from the optimized simulation, in the bottom trace the effects are removed. In this figure two of three operation modes of the operation can be seen clearly.
The first one is in the time span from 47.4 ms to 69.7 ms of optimized trace that corresponds to the time span from 24.8 s to 26.3 s of the last trace. This equals to a time reduction of 98.51%. During this time the input from the Gazebo simulation was only the changing field of the removing NFC reader. This value is constant for any time step. As the rest of the sensor is idle, the simulation is only run for a short period of time to check if any values are changing in an unexpected way, after that the simulation is skipped and the rest of the changes are approximated. This can be seen in a thin line in the *advanceTime* line.
In contrast to that, the first 1.2 ms of the optimized trace correspond to the first 6.3 s of the last trace. This equals to a time reduction of 99.98%. In this time no RF field is detected and the sensor is idle. In this phase the simulation is only started if the *time debt* of Figure 9 reaches a predefined value to update the change rates due to non-linear effects. During this phase most changes are approximated. The *advanceTime* line shows a thick line while this phase lasts.

The third operation mode is seen if the *advanceTime* line is not elevated. Figure 13 shows a section of Figure 12 where the sensor is not idle. This phase starts from 43.5 ms of the optimized trace and has a duration until 43.74 ms. This corresponds to the time span from 14 250 ms to 14 250.24 ms. In this time the sensor is receiving data that needs to be processed. Therefore the sensor is not idle and the effects can not be estimated. Here the full simulation is performed and no time reduction occurs.

The values of the simulation time reduction highly depend on the processor state and the simulated use case.

Chapter 8. Model-based Design of Secured Power Aware Smart Sensors

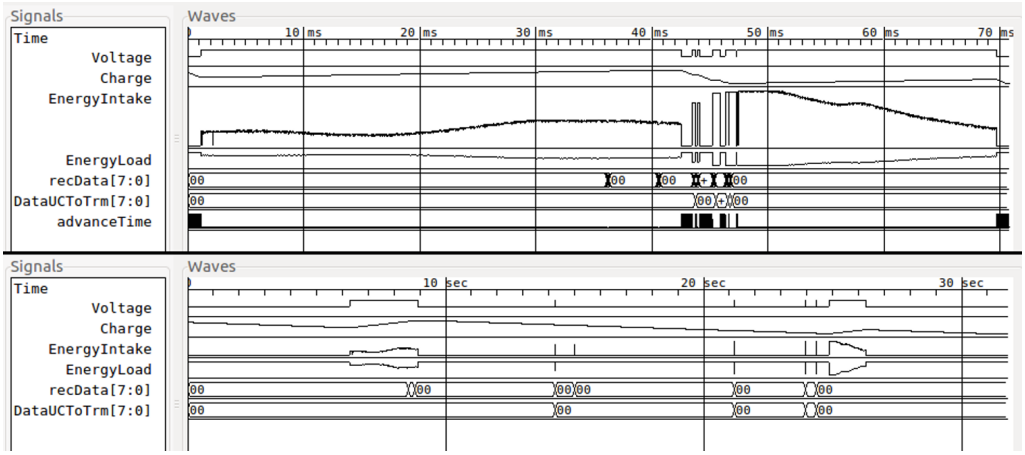

Fig. 12 Effects that result from the runtime optimization by skipping simulation steps. The simulation time is compressed if the sensor is idle and the effects can be estimated.

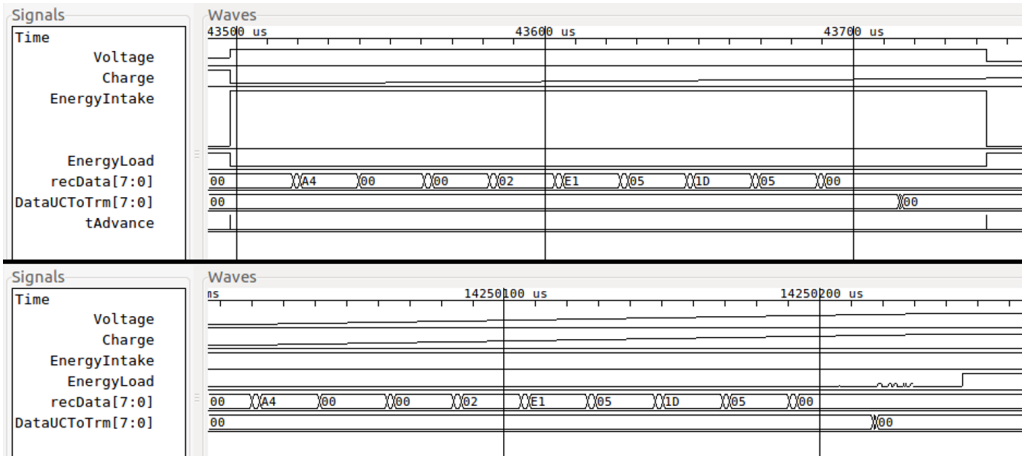

Fig. 13 Third operation mode of the optimization. If the sensor is not idle, no compression occurs.

7 Security of Smart Sensors

When developing new sensors, not only the efficiency of them is important, they furthermore should be secure. The security should not only care about the data that is generated by the sensor, but also the sensors should not endanger systems they are connected to.

Any security solution for smart sensors needs to consider the following six points:

- The sensor has limited resources in terms of energy and processing power.
- The overhead in memory usage, computational effort, and data transmission sizes that are imposed due to security considerations can be large in comparison to the payload data.

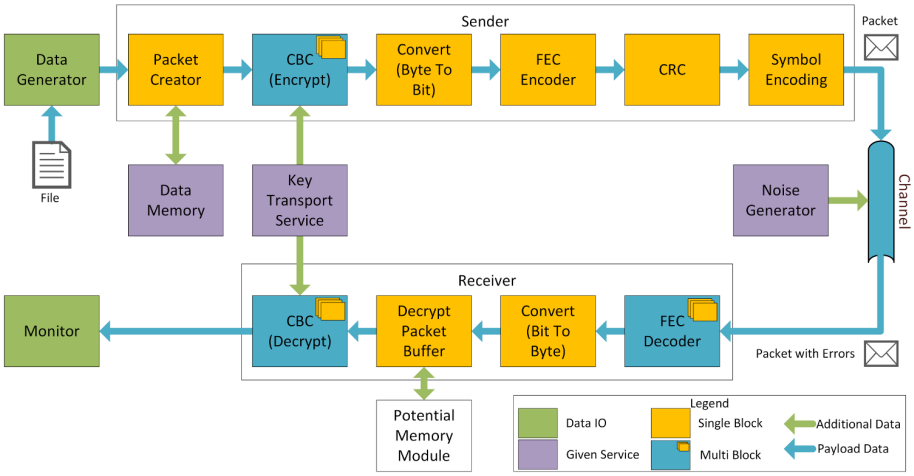

Fig. 14 Concept to simulate the JEEC approach by Ulz et al. [24].

- To provide security to a sensor network every sensor needs to perform the security relevant operations.
- The security features should be easy to use by untrained end users.
- The end user needs to trust the security features of the sensor.
- Dedicated hardware that performs the security operations is better, but needs more energy and can be costly.

7.1 Data Security

The sensor data needs to be protected from unauthorized access. Otherwise this data can disclose company secrets. Furthermore, if the sensor data can be replicated, control mechanisms can be fooled to perform harmful operations that can endanger the facility or human lives. In the case of the STUXNET attack [9] the attacker has learned the behaviour of the system by observing the sensor data. After the learning phase the attacker manipulated the sensor's data to turbines of a nuclear enrichment plant while it was hiding its doings by displaying normal looking data to the next layer of controls.
This attack showed that the sensor data needs to be protected. To be able to read (or guess) the sensor data can enable an attacker to manipulate the processes they observe.

To protect the gathered data in a sensor network from unauthorized access, the data needs to be encrypted. As Ulz et al. showed, such measures increase the severity of introduced bit errors [24]. To still be able to communicate correctly, a forward error correction scheme can be used on the encrypted data. Using this method, bit errors can be detected before decrypting, thus increasing the resilience against bit errors of the communication. Ulz et al. also suggested a solution to this problem which they called Joint Encryption and Error Correction (JEEC).

Chapter 8. Model-based Design of Secured Power Aware Smart Sensors

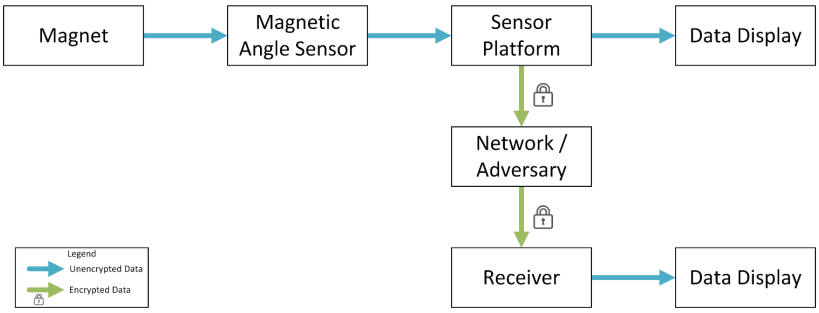

Fig. 15 Concept for testing the secured data transmission of Ulz et al. [24].

To simulate the security of the data during transmission, another simulation has been created. Figure 14 shows the concept of this simulation. Here the green fields represent blocks where the data is not secured. The yellow and dark blue fields represent steps for the data processing, where the dark blue describes that this process consists of many subprocesses. The light blue fields are considered given in this simulation.
The data to be transmitted is provided by a file. The top row shows the process the sender takes before the data is sent. During this process the data is chunked into packets, encrypted, a forward error correcting (FEC) code is added, and a checksum is calculated. This packet is then sent via a channel. This channel can see the transmitted data and introduce bit errors. Before the receiver can use the data, it needs to be processed again. This process, shown in the bottom row, consists of the recovering of flipped bits using the FEC and decrypting.

Figure 15 shows a concept to test the JEEC approach. Here a magnetic angle sensor is stimulated by a magnet. The sensor data (blue arrows) is then sent to the smart sensor platform where it is processed. The original data gets displayed for later comparison. Additionally, the sensor data gets encrypted and encoded using a FEC scheme. This packed data is then sent via a network to the receiver (green arrows with locks). The adversary is in the network and can manipulate the data. The receiver tries to decode the data. If this succeeds the plain data is displayed to be compared against the original one.

7.2 System Security

Recently another kind of attack using sensors has been reported. In this scenario a large number of sensors connected to a network is used to perform a DDoS (Distributed Denial of Service) attack. In this type of attack the sensors firmware is altered to send messages through the network to a common destination. Using a large number of manipulated sensors, the common destination receives more data than it can handle. This blocks the recipients capabilities to perform its normal operations.

To use the sensors for such malicious purposes, their firmware needs to be changed. Ulz et al. presented an approach to exclude sensors from the sensor network that use unauthorized firmware [22]. This is done in a two-layer system. in the first step the firmware is checked by an on-board security module. If this check verifies that the software is from an trusted source, the sensor's network stack is released. Now the firmware version can be checked at a trusted backend server. This verifies that the sensor is running the newest version of the firmware.

8 Conclusions

This chapter presents an approach to design secured power aware smart sensors using hardware models. To be able to use this model-based design, the model parameters need to be known. Therefore, we discussed the methods for gathering such data and how the data can be simplified to be used in an abstracted hardware model. In addition to the model, use cases for the new sensor need to be known. Here a simple simulation can be performed that generates the inputs necessary for the hardware simulation. If the use case extends beyond one sensor, the gathered data can be abstracted again and included in higher layer simulations. Additionally, we presented the main phases of such simulations and mentioned possibilities for runtime optimizations. Finally, we discussed the possibilities of attacks on and with the sensors and mentioned methods to increase the difficulty of a successful attack. Additionally, the concepts to secure the data against adversaries has also been simulated and tested.

As the simulation should resemble a generic smart sensor, some details are lost at every layer of abstraction of the models. This loss in detail is necessary to create usable simulations.

The main focus of this chapter is the creation of the simulation of such sensor systems. A detailed description on the model generation and the simulation steps are given.

Using these simulations it is possible to analyse the effects of changes to the sensor system, the communication protocols, or the routines in the environment. This can be used to optimize the energy consumption of each individual components as well as the energy consumption of the complete environment.

Acknowledgements This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 692480. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Netherlands, Spain, Austria, Belgium, Slovakia.

IoSense is funded under the agreement number 853326 by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2016 and May 2019. More information https://iktderzukunft.at/en/

We want to thank Infineon Technologies and especially Rainer Matischek for providing us the security controllers used in the system and for their support that helped creating the prototypes and simulations.

References

1. Accelera: SystemC. http://accellera.org/downloads/standards/systemc (2000). Last accessed on Jan 17, 2017
2. Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J.A., Invernizzi, L., Kallitsis, M., et al.: Understanding the mirai botnet. In: USENIX Security Symposium (2017)
3. Bouchhima, F., Briere, M., Nicolescu, G., Abid, M., Aboulhamid, E.: A SystemC/simulink co-simulation framework for continuous/discrete-events simulation. In: 2006 IEEE International Behavioral Modeling and Simulation Workshop. Institute of Electrical and Electronics Engineers (IEEE) (2006). DOI 10.1109/bmas.2006.283461. URL http://dx.doi.org/10.1109/BMAS.2006.283461
4. Campbell, R.J.: Cybersecurity issues for the bulk power system (2015)
5. Chen, G., Fojtik, M., Kim, D., Fick, D., Park, J., Seok, M., Chen, M.T., Foo, Z., Sylvester, D., Blaauw, D.: Millimeter-Scale Nearly Perpetual Sensor System with Stacked Battery and Solar Cells. In: 2010 IEEE International Solid-State Circuits Conference - (ISSCC). IEEE (2010). DOI 10.1109/isscc.2010.5433921
6. Haase, J., Meyer, D., Eckert, M., Klauer, B.: Wireless sensor/actuator device configuration by nfc. In: Industrial Technology (ICIT), 2016 IEEE International Conference on, pp. 1336–1340. IEEE (2016)
7. Huang, K., Bacivarov, I., Hugelshofer, F., Thiele, L.: Scalably distributed SystemC simulation for embedded applications. In: 2008 International Symposium on Industrial Embedded Systems. Institute of Electrical and Electronics Engineers (IEEE) (2008). DOI 10.1109/sies.2008.4577715. URL https://doi.org/10.1109%2Fsies.2008.4577715
8. Kansal, A., Hsu, J., Zahedi, S., Srivastava, M.B.: Power Management in Energy Harvesting Sensor Networks. ACM Trans. Embed. Comput. Syst. **6**(4) (2007). DOI 10.1145/1274858.1274870
9. Langner, R.: Stuxnet: Dissecting a cyberwarfare weapon. IEEE Security & Privacy **9**(3), 49–51 (2011)
10. Lee, W.S., Son, W.I., Oh, K.S., Yu, J.W.: Contactless energy transfer systems using antiparallel resonant loops. IEEE Transactions on Industrial Electronics **60**(1), 350–359 (2013). DOI 10.1109/tie.2011.2177611. URL https://doi.org/10.1109%2Ftie.2011.2177611
11. Martin, D., Wilsey, P., Hoekstra, R., Keiter, E., Hutchinson, S., Russo, T., Waters, L.: Integrating multiple parallel simulation engines for mixed-technology parallel simulation. In: Proceedings 35th Annual Simulation Symposium. SS 2002. Institute of Electrical and Electronics Engineers (IEEE) (2002). DOI 10.1109/simsym.2002.1000082. URL https://doi.org/10.1109%2Fsimsym.2002.1000082
12. Mathworks: Get Started with Gazebo and a Simulated TurtleBot. https://de.mathworks.com/help/robotics/examples/get-started-with-gazebo-and-a-simulated-turtlebot.html (2016). Last accessed on Jan 03, 2017
13. Mueller-Gritschneder, D., Lu, K., Wallander, E., Greim, M., Schlichtmann, U.: A virtual prototyping platform for real-time systems with a case study for a two-wheeled robot. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013. EDAA (2013). DOI 10.7873/date.2013.274. URL https://doi.org/10.7873%2Fdate.2013.274
14. Open Source Robotics Foundation: Gazebo simulator. http://www.gazebosim.org (2004). Last accessed on Jan 03, 2017
15. Panda, P.R.: SystemC - Amodelling platform supporting multiple design abstractions. In: Proceedings of the 14th international symposium on Systems synthesis - ISSS. Association for Computing Machinery (ACM) (2001). DOI 10.1145/500001.500018. URL https://doi.org/10.1145%2F500001.500018
16. Pieber, T.W., Ulz, T., Steger, C.: Systemc test case generation with the gazebo simulator. In: Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH,, pp. 65–72. INSTICC, SciTePress (2017). DOI 10.5220/0006404800650072

17. Pieber, T.W., Ulz, T., Steger, C., Matischek, R.: Hardware secured, password-based authentication for smart sensors for the industrial internet of things. In: International Conference on Network and System Security, pp. 632–642. Springer (2017)
18. Possadas, H., Adamez, J.A., Villar, E., Blasco, F., Escuder, F.: RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model. Design Automation for Embedded Systems (2005). DOI 10.1007/s10617-006-9725-1
19. Rahimi, M., Shah, H., Sukhatme, G., Heideman, J., Estrin, D.: Studying the Feasibility of Energy Harvesting in a Mobile Sensor Network. In: 2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422). IEEE. DOI 10.1109/robot.2003.1241567
20. Sogorb, T., Llario, J.V., Pelegri, J., Lajara, R., Alberola, J.: Studying the feasibility of energy harvesting from broadcast rf station for wsn. In: Instrumentation and Measurement Technology Conference Proceedings, 2008. IMTC 2008. IEEE, pp. 1360–1363. IEEE (2008)
21. Ulz, T., Pieber, T., Steger, C., Haas, S., Matischek, R.: Sneakernet on wheels: Trustworthy nfc-based robot to machine communication. In: RFID Technology & Application (RFID-TA), 2017 IEEE International Conference on, pp. 260–265. IEEE (2017)
22. Ulz, T., Pieber, T., Steger, C., Haas, S., Matischek, R., Bock, H.: Hardware-secured configuration and two-layer attestation architecture for smart sensors. In: Digital System Design (DSD), 2017 Euromicro Conference on, pp. 229–236. IEEE (2017)
23. Ulz, T., Pieber, T., Steger, C., Lesjak, C., Bock, H., Matischek, R.: Secureconfig: Nfc and qr-code based hybrid approach for smart sensor configuration. In: RFID (RFID), 2017 IEEE International Conference on, pp. 41–46. IEEE (2017)
24. Ulz, T., Pieber, T., Steger, C., Matischek, R., Bock, H.: Towards trustworthy data in networked control systems: A hardware-based approach. In: Emerging Technologies and Factory Automation (ETFA), 2017 22nd IEEE International Conference on, pp. 1–8. IEEE (2017)
25. Yan, R., Sun, H., Qian, Y.: Energy-aware sensor node design with its application in wireless sensor networks. IEEE Transactions on Instrumentation and Measurement **62**(5), 1183–1191 (2013). DOI 10.1109/tim.2013.2245181. URL `https://doi.org/10.1109/tim.2013.2245181`
26. Zamora, I., Lopez, N.G., Vilches, V.M., Cordero, A.H.: Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo. arXiv preprint arXiv:1608.05742 (2016)

Authors' Biographies

Chapter 8. Model-based Design of Secured Power Aware Smart Sensors

Thomas W. Pieber received his master's degree (M.Sc.) in Information and Computer Engineering from Graz University of Technology in 2016. The focus of his studies included Embedded Systems, Mobile Computing, and Robotics. Currently he is a Ph.D. student in Information and Computer Engineering for the Institute for Technical Informatics, Graz University of Technology. His research currently focuses on energy efficiency of IoT devices.

Thomas Ulz received his master's degrees (M.Sc.) in Information and Computer Engineering as well as in Computer Science from Graz University of Technology, both in 2016. The focus of his studies included Security, Embedded Systems, Robotics, and Machine Learning. Currently, he is a Ph.D. student in Information and Computer Engineering at the Institute for Technical Informatics, Graz University of Technology. His research currently focuses on security aspects of IoT devices.

Christian Steger received the Dipl.-Ing. degree (M.Sc.) in 1990, and the Dr. techn. degree (Ph.D.) in electrical engineering from Graz University of Technology, Austria, in 1995, respectively. He graduated from the Export, International Management and Marketing course at Karl-Franzens-University of Graz in June 1993 and completed the Entrepreneurship Development Program at MIT Sloan School of Management in Boston in 2010. He is strategy board member of the Virtual Vehicle Competence Center (ViF, COMET K2) in Graz, Austria. Since 1992 he has been Assistant Professor at the Institute for Technical Informatics, Graz University of Technology were he heads the HW/SW codesign group at the Institute for Technical Informatics.

Towards Continuous Sensor Operation: Modelling a Secured Smart Sensor in a Sparse Network Operated by Energy Harvesting

Thomas W. Pieber, Benjamin Mößlang, Thomas Ulz and Christian Steger
Institute for Technical Informatics, Graz University of Technology, Inffeldgasse 16/I, Graz, Austria
{*thomas.pieber, thomas.ulz, steger*}*@tugraz.at,*
benjamin.moesslang@student.tugraz.at

Keywords: Energy Harvesting, NFC, Robotics, Smart Factory, Smart Home, Smart Sensors, Sparse Networks

Abstract: In modern society sensors are omnipresent. They gather information about their environment in order to optimize production flows, minimize energy usage, learn about the environment, or maximize the owner's comfort. To achieve the desired goal in already existing buildings, sensors are introduced afterwards. These sensors might not be able to connect to a sensor network because of obstacles or user policies. If this happens, other mechanisms to create a network to gather the data need to be found. Additionally, these sensors should last for a long period and are therefore probably powered using energy harvesting methods. In this paper we present an approach for simulating the charging process of such sensors and connecting them to a network using mobile communication partners.

1 Introduction

In our society we use sensors to automatically gather data for almost every aspect of our environment. In some applications the sensors cannot connect to a network, either it is not feasible to build infrastructure to connect the sensors (1) or they are not allowed to join an existing network (2).

The first use case is most likely to arise in a sparse sensor network such as when monitoring a wide area or if obstacles such as buildings influence the communication channel. Furthermore, this can arise in an Industry 4.0 setting where sensors can be added at any time and due to insufficient wireless coverage, interferences, obstacles, or policies the sensors cannot connect to the local network.

In the second case the sensors can collect data that needs to be handled confidentially and are therefore not allowed to be transmitted over a long-range wireless communication channel. This can also happen in an Industry 4.0 scenario.

For these use cases mobile communication partners (nodes) with additional computational power and further capabilities can be introduced to connect the sensor nodes to the data sink. These mobile partners can be a worker or robot in a factory, a home owner in his house, or an employee of the city. These communication partners then need to estimate the urgency of the collected data, the memory usage of the sensor, and the energy level of the sensor nodes they should connect to the network. This results in periodical visits from the mobile node at any sensor. These visits can furthermore be used to prolong the sensor's operational time without much effort.

Additionally, when the nodes are visited and their data is collected by the mobile node it may also be possible to configure the sensors behaviour to account for changing needs of the owner. To mitigate possible threats that come from unauthorized personnel changing the configurations, the sensors need to be secured.

There are many proposed solutions that connect the sensors to mobile nodes (eg. (Marta and Cardei, 2009; Ye et al., 2002; Kim et al., 2003)). Most of these use traditional radio frequency communication to communicate the distance to the mobile node. Transporting energy in addition to the data requires other communication technologies such as Near Field Communication (NFC). In this work we examine the possibility of using NFC-enabled robots, or NFC-enabled smartphones to charge, read out, and configure sensor nodes which cannot be connected in a traditional way. This should happen while the data is gathered reliably and secured.

The design of such sensor nodes poses many complex questions such as: "Where is the energy needed the most?", "How much energy can be saved by reducing the sample frequency?", or "How can the energy be used more effectively?". To answer these

Original work published in Proceedings of the 9th International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2019), pages 57-64. SCITEPRESS - Science and Technology Publications, 2019.

questions, simulations can be used to gain insights in this complex topic. These simulations focus on the detailed description of the sensor system and try to give detailed answers. This however neglects other major questions such as: “How does the designed energy harvesting system perform?” or “How does the environment affect the sensor communication?”.

This publication focuses on the creation of a simulation that is also able to answer questions like these. To do this we created a simulation that focuses on the gathering of time insensitive information using sensor nodes in an environment that does not allow long range wireless communication. Such scenario might arise when measuring the temperature in an automated warehouse, the air humidity of a basement, or the air quality in a city. In these scenarios, some agents (robots or humans) constantly move through the environment performing operations. We propose that they can also collect the information from the sensor nodes, keep them operational and adapt the sensor nodes behaviour while these agents are performing their jobs.
To effectively use such a method requires the prior knowledge of the sensors energy consumption. This is necessary as the sensors operational lifetime is limited by the stored energy. The periodic visits should happen shortly before the internal memory is full or the energy level drops below a certain level to optimally use the mobile agent’s time. Thus, this publication explores a method to estimate the energy consumption of a smart sensor and the possibilities that energy harvesting offers to prolong the sensors lifetime.

The remainder of this paper is structured as follows: In Section II related work is described. This section is split to emphasize the energy harvesting, the energy usage estimation and the mobile data collection. The theory and approach of our experiments can be found in Section III. Here we describe the hardware prototypes, experiments and the performed simulations. Section IV is dedicated to the measurements taken with the prototypes and compares it to the results of the simulations. Section V contains ideas on how to improve on the findings of this paper. This paper concludes in Section VI.

2 Related Work

2.1 Energy Harvesting

Prolonging a battery powered sensor’s lifetime is the goal of many publications. Most of the publications include mechanisms for harvesting energy from the surrounding. These harvesting methods include the use of solar cells (Chen et al., 2010) or antennas (Pinuela et al., 2013) to use electromagnetic radiation, devices to convert heat gradients (Dziurdzia and Stepien, 2011) to electrical energy, and mechanisms to utilize mechanical energy (Choi et al., 2006).

All those methods are built to support the sensor with a steady (or at least calculable) amount of energy. Thus, the sensors need only small energy storage capacities to dampen energy fluctuations. (Kansal and Srivastava, 2003; Kansal et al., 2007; Chen et al., 2010; Tan and Panda, 2011).

J. Gummeson et al. (Gummeson et al., 2014) developed a small worn sensory device. This device is embedded inside a ring which limits the size available for energy storage. In their solution to this problem they used NFC to recharge the internal storage whenever the user reaches for an NFC-enabled smartphone. A similar approach can be used to operate larger devices. In the case of a smart sensor the energy storage can be made significantly larger, allowing for longer operation between the recharging occurs. At the same time larger antennas and more powerful NFC-readers can reduce the time needed for recharging the sensor. We also plan to use NFC to charge the sensor. In contrast to the work of Gummeson et al. we want to power a system that requires more energy.

A study from M. Rahimi et al. (Rahimi et al., 2003) investigates the feasibility of mobile nodes in order to provide energy to a sensor network. In this study the authors show promising results of their prototypical tests. This work focuses on the mechanisms for searching for energy in an environment and how the robots split the servicing (charging) of the sensor nodes. They focus their research on quantifying the power consumption of the network and specifying if the network is sustainable.
In contrast to this research we focus on the combination of energy transfer to the sensor nodes and the simultaneous data collection performed by mobile nodes.

Chen et al. (Chen et al., 2010) proposed and demonstrated a nearly self sustaining micro sensor that uses solar cells to generate the needed energy for sensing capacitance and temperature . This approach to build a self-sustaining sensor network seems promising but induces the need that the sensors are subject to a sufficient light source which might not be given in many scenarios. The work of Kansal et al. (Kansal et al., 2007) described power management techniques that can be used and they described how such mechanism can be implemented with respect to a known model of the desired energy source. That can be used to improve the work of Chen et al. (Chen

et al., 2010). Additionally, such a model can be used in the proposed model to optimize the sensor usage.

In this paper we try to use NFC as communication technique to transmit data between a sensor and a mobile node. In this approach a robot is acting as the mobile node. This robot creates the connection to the infrastructure, thus acting as a slow and random link. The use of NFC furthermore allows us to transmit power to the sensor supporting it for high power operations and keeping it operational.

2.2 Energy Usage Estimation

The estimation of the energy consumption of sensors is an ongoing research. There are approaches to minimize the energy consumption of sensor nodes based on their specific energy levels and that take into account the energy levels of the surrounding nodes (Yan et al., 2013).

Other researchers such as Halgamuge et al. (Halgamuge et al., 2009) generated a model of a sensor's behaviour and try to estimate the energy consumption of the sensor based on the information of the behaviour.

To get better results than pure estimation of the energy consumption, we decided to create a research prototype of a low-power sensor on which the energy consumption can be measured.

2.3 Mobile Communication Partners

The idea of using mobile nodes in order to connect a wide spread sensor network has been explored widely in the existing literature. Most of these solutions use erratic moving partners (such as animals in their habitat who are equipped with a sensor node) to try to connect all of the stationary nodes(Shah et al., 2003; Ulz et al., 2017a; Rahimi et al., 2003).

The approach developed by Ulz et al. (Ulz et al., 2017a) to connect industrial machines using robots as links can be used to calculate the sensor node that should be visited next.

In the work of Shah et. al. (Shah et al., 2003) a multi-layer network with mobile nodes to connect sensors with each other and with the data sinks was proposed and explained. One of their main goals was to minimize the sensors memory to decrease energy demand. In their studies they proposed to mount the mobile nodes on animals, roaming through their habitat. In their assumption the mobile nodes (MULEs) are performing a random walk and stumble upon the sensors is not applicable to our use case. With that we can simplify many of the calculations done in order to get a reasonable memory size. Furthermore, as the mobile partner can directly communicate with the sensor, the sensor can suggest a return time for the mobile partner to improve memory usage and sensor lifetime.

Rahimi et. al. (Rahimi et al., 2003) describe an approach for energy harvesting and distributing the energy in a wireless sensor network (WSN) with the help of mobile autonomous robots. In their approach the robot moves through the observed area and finds a spot with enough available solar energy to charge the battery. The robot then moves towards the sensor nodes which need the energy and charges them. With this approach also sensors that can not harvest enough energy to sustain themselves can be operated using the delivered energy. This approach furthermore increases the lifespan of the rest of the sensors as they are provided with more energy than they could harvest on their own. In our approach the sensors are not only sustained by the mobile robot, but also their data is collected. This cuts the energy demand for transmitting the gathered data, allowing the sensors to operate longer.

In our presented approach we use controllable means of transportation in order to efficiently collect all gathered data, update configurations of the sensors, and charge the batteries. This controlled data collection can also provide means to predict the arrival of new data and the possibility to have information on the timeliness of the data.

3 Approach

Many systems, designed to be used as a sensor for a sensor network, use batteries, wired electrical connections, or continuous energy harvesting methods as their main power source. To utilize short bursts of energy as power source, the energy must be received and the excess must be stored in a usable manner. To do this, the energy is stored using accumulators or capacities. As we try to combine the transport of data and the delivery of energy, NFC technology, and thus energy bursts, are examined.

To use NFC as means of energy transport, the circuitry of the smart sensor must allow the extraction of excess energy of the NFC field. Furthermore, the excess energy must be directed to charge either a capacitor or an accumulator. This is done by extracting electrical energy from the electromagnetic field. The generated AC (alternating current) voltage is then rectified and the voltage and current are controlled to protect the sensor. To be more efficient, the circuitry can include means of distinguishing between operating the sensor from the stored energy and operating

it from the NFC field and storing the excess energy. This switching of operation mode can be performed by the sensor's main controller.

To store the energy different solutions can be used. For this domain the most useful solutions are accumulators or capacitors. The energy density of capacitors is lower in comparison to accumulators. This means that capacitors can hold less energy. In contrast to that the power density of capacitors is larger. This allows capacitors to store and draw energy faster than equally sized accumulators (Zhang et al., 2013).

We decided to use a super-capacitor based development board as basis for the research prototype measuring the energy provisioning system (henceforth Prototype A).In addition to the prototype to measure the energy provisioning system, we created a research prototype on which the energy consumption of the sensor components can be measured (henceforth Prototype B).

Using these research prototypes, we gather data to set up simulations that can represent the interactions between the sensors and the mobile agents, as well as help in gathering data to optimize the agent's visiting schedule. Here a simulation approach by Pieber et al. (Pieber et al., 2017a; Pieber et al., 2017b) can be used to connect the simulation of a smart sensor system to a simulation that is more capable of simulating the interaction between an agent and the sensor.

3.1 Research Prototypes

3.1.1 Prototype Overview

An MSP430FR5969 development board (TI, 2014) is chosen as a basis for Prototype A. This board includes a super-capacitor as energy source, a microcontroller tailored for low power use, and a temperature sensor. It is then extended with a custom made PCB to connect an NFC interface (ams AG, 2006), capable of handing energy to the host system. The extension PCB can be configured using jumpers to allow different communication channels. This PCB furthermore features a simple power management system that limits the power that can reach the super-capacitor and can switch between the available power sources such that the capacitor can be charged when an NFC field is present. A circuit plan of the prototypical PCB is shown in Figure 1.

Prototype B consists of three parts. The *Energy Measurement Unit* (EMU, Testbench), a control computer, and the *Smart Sensor* itself. As basis for the EMU another PCB was designed. This is fitted to an MSP340FR5969 board that gathers the data and acts as a bridge between the measurement unit and the control computer.
The smart sensor consists of a microcontroller, interface ports, additional memory, an NFC interface, and a security co-processor. Each of these components is supplied by an energy channel coming from the measuring testbench. Additionally, most components can be cut from the power supply to reduce the energy demand of the sensor.
The testbench is controlled by a LabVIEW computer simulating the energy provisioning system. This is done by reading the energy demand of the sensor and setting the supply voltage according to the capabilities of the simulated provisioning system. In addition to the supply voltage the LabVIEW script also sets the digital potentiometers to adapt the gain of the current sensors. Using the information about the current drawn by the sensor components and the voltage of the system, an energy profile can be created that can be used to create simulations describing similar sensors. The design for Prototype B is shown in Figure 2.

3.1.2 Prototype Details

When the NFC antenna of Prototype A is subjected to an NFC field, a DC (direct current) voltage is generated. This voltage is represented as *NFC DC* in Figure 1. If this voltage is larger than the voltage at the capacitor *C* connected to *CHARGE+* the controller switches the analog-switch connected to *GPIO* such that the capacitor is connected to the voltage source via the resistor (*723* Ω). This controls the current that can pass through the capacitor. If the voltage difference between *CHARGE+* and *VCC* is smaller than a threshold, the switch is flipped and the resistor is short-circuited. This leaves the capacitor connected to *VCC* via the Schottky diode *D*. With this the current can not flow to the capacitor and it will not be charged any more. If the voltage of *NFC DC* is smaller than the voltage of the capacitor at *VCC*, the controller is powered from the capacitor. Should the voltage at *VCC* drop below a threshold, the controller switches in a low-power mode and waits for a voltage increase at *VCC* to start the charging again.
The voltages and currents that are present in the system can be measured at the jumper pins on the PCB. The most interesting values are the voltage at the capacitor, the voltage that reaches the sensor and the current that is drawn by the sensor.

The three parts of Prototype B are shown in Figure 2. The *Smart Sensor*, the *Testbench*, and the *Computer/Control* are represented as the boxes that combine the necessary elements.
The *Control* is located at a LabVIEW computer. This computer receives the measurements of the testbench

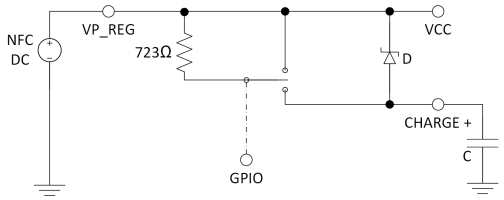

Figure 1: Circuit of the energy provisioning system of Prototype A.

and sets the control signals for the testbench such that a specified energy provisioning behaviour is reached. The *Testbench* receives these signals and controls the variable voltage source and the digital potentiometers according to the control signals. It furthermore relays the measurement values of the current sensors to the *Control* computer. The *Testbench* has seven variable gain current sensors that can be used to observe the behaviour of the device under test.
The device under test - in this case the *Smart Sensor* - consists of six components, and therefore uses six measurement channels. The control unit of the sensor is an *Ambiq Micro Appollo 2 MCU*. This controller is connected via an I^2C bus to an *Optiga Trust X Security Co-Processor*. Another separate I^2C bus connects to an additional *FRAM* module as well as to an *NFC Interface*. Additionally, two *External Ports* are connected via IO pins. At these interfaces, different expansion modules can be connected. Using the IO pins, I^2C or *SPI* buses can be simulated to communicate with sensors, actuators, or other controllers.
Using *Load Switches*, the controller is able to cut different components off the energy supply to reduce the energy demand of the smart sensor as far as possible. This setup allows the measurement of the energy demand of each component of a smart sensor in a flexible way.

3.2 Simulation

The data generated from the prototypes is fed into simulations describing similar sensor systems. These simulations are necessary, as we are interested in the behaviour of the sensor in combination with different systems.
To get a more abstract system description of a smart sensor the gathered data needs to be generalized. Thus deviations in the results of the simulations from the measurements are expected.
To describe the sensor system electrically, a simulation using a SPICE program is created that represents the smart sensor. Here simplified but usable parameters can be extracted that are used in the calculations for the energy consumption of the smart sensor.

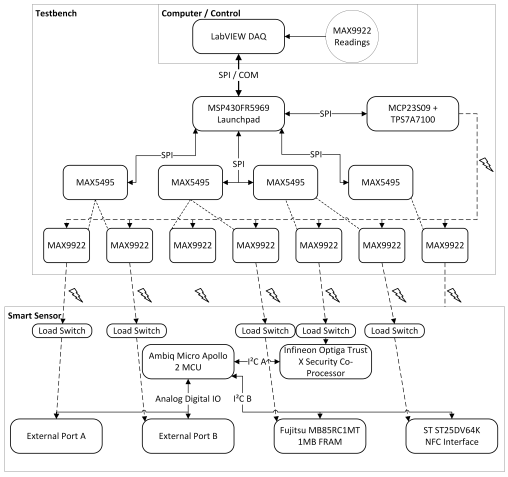

Figure 2: Design of Prototype B.

The gathered parameters are then used to describe the electrical behaviour of the sensor is subsequent simulations.

The simulation of the sensor itself is written in SystemC as it allows the description of the entire system at different levels of abstraction. This is especially useful as the accurate simulation of a complex system impedes the simulation performance. This is counteracted by performing simulations on a more abstract level.
The environment is simulated using the Gazebo simulator, a simulation tool commonly used for robotic purposes. These two simulations are connected using a method developed by Pieber et al. (Pieber et al., 2017a). In Figure 3 such a simulation run can be seen. The robot interacts with the sensor, charging it, collecting data, and possibly reconfiguring it to alter the sensors behaviour. Using this simulation technique, the sensor simulation gets the stimuli from the environment and reacts according to it. This allows the quick creation of new test cases and stimuli, as well as the automatic evaluation of sensor responses.

Within the Gazebo environment, it is possible to align the antennas in various orientations to each other. Additionally, it is possible to introduce extra noise (such as noise from nearby communications or multi-path signal propagation) to the communication. This allows to find the answers to the questions asked in Section 1.

Summarized, this means that the measurements of the prototypes are used to generate a SPICE simulation. Parameters are extracted from this simulation that can be used to describe the electrical behaviour of the sensor. This description is done in SystemC. To

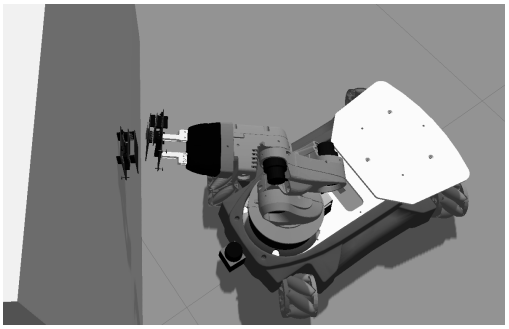

Figure 3: Simulation of an interaction between robot and sensor.

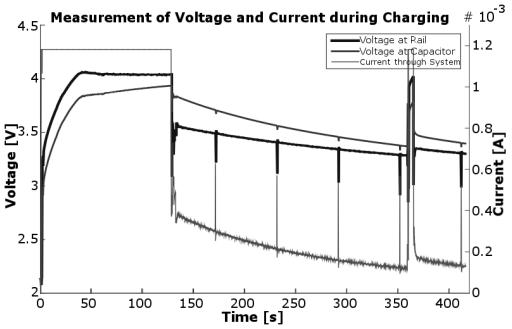

Figure 4: Measurement of the charging of Prototype A.

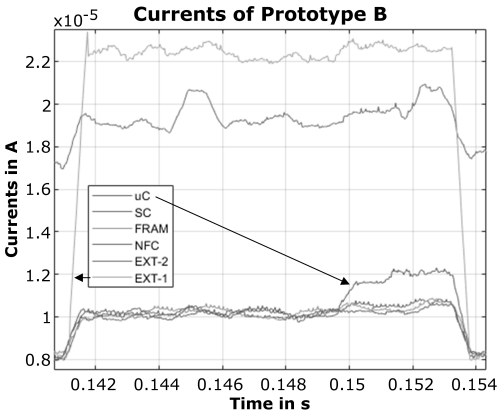

Figure 5: Current flow during measurements at Prototype B.

subject the SystemC simulation to stimuli, the Gazebo simulator is used. This is done in order to quickly change the alignment of the NFC antennas and to introduce additional noise to the system.

This simulation yields data about the interaction of the smart sensor with the environment in different situations. Furthermore, information about the energy usage during operation as well as information about the memory usage of the stored values are created. This information, in connection with information about the type of generated data, can then be used by a mobile node to calculate the need to visit the sensor node to collect data and recharge it.
If multiple sensors are connected in this way, a sensor network is created using the robots as links between themselves and the infrastructure. This method was described by Ulz et al. (Ulz et al., 2017a) as a "Sneakernet on Wheels".

4 Measurements and Results

Prototype A is designed to measure the energy harvesting capabilities of the NFC connection. A typical measurement of this prototype can be seen in Figure 4. In this measurement the capacitor was charged for approximately 120 sec. During this time the voltage at the capacitor (middle line) reached the nominal voltage. After that the NFC field was switched off and the voltage at the rail (bold line) drops below the voltage at the capacitor. The current through the sensor (fine line) also drops significantly as the controller switches to a low-power mode. After that the controller awakes every 60 sec to take a measurement. At this times the current rises and the voltage at the rail drops as the capacitor and diode pose a resistance to the current. Again at 360 sec the NFC field is switched on briefly.

The operation of the smart sensor is measured using Prototype B. In Figure 5 a sample measurement is shown. In this example the controller switches on an external sensor at port *EXT-1*. After that, it waits until the sensor has initialized itself and starts the measurement. In this test, the controller finally saves the gathered data and shuts down the sensor.
The microcontroller (μC) current rises at the start of the measurement as the sensor module starts its operation. During the communication with the sensor and the subsequent storing of the gathered values the power consumption and therefore the needed current rises further.
The continuous high current at *EXT2* is generated by other sensor hardware. Furthermore, the current usage by the Security Controller (*SC*), and the *FRAM* and *NFC* modules are increased at the start of the measurement routine.

Using the available data and the data from the SPICE simulations, a model can be created that shows the energy consumption of a generic smart sensor. Figure 6 shows the results of a simulation using the sensor model. In this simulation the charging be-

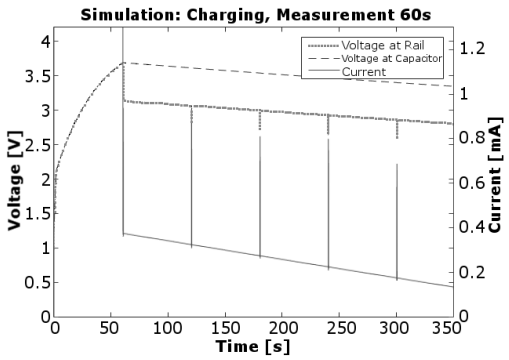

Figure 6: Simulation of the sensor charging.

haviour of the sensor was simulated. This behaviour is comparable to the measurements of Prototype A, thus a valid simulation has been created this way.

These results help in predicting the time it takes to charge one sensor using NFC energy harvesting and can furthermore help in estimating the time it needs to discharge the smart sensors energy storage.

In summary these results show that the charging of a smart sensor using NFC is possible. To charge the 0.5 F capacity approx. 2 minutes are needed. This also depends on external parameters such as the alignment of the antennas. The simulation of the sensor furthermore shows that the sensors operational time primarily depends on its duty cycle. While the sensor can be operated for approximately one day when performing measurements every hour, the lifetime is cut to about seven hours when measuring every minute.

5 Future Work

Using the prototypes we can perform measurements of security relevant operations. As Prototype B also features a security coprocessor, we want to perform experiments determining the difference in energy usage during cryptographic operations. These experiments can show the requirements smart sensors and the energy provisioning systems need to fulfil to enable secured data handling. We are planning to take measurements of user authentication algorithms on smart sensors such as the one proposed by Pieber et al. (Pieber et al., 2017c), and testing data transmission protocols that secure the transmitted data using encryption and forward error correction such as the one proposed by Ulz et al. (Ulz et al., 2017b).

6 Conclusion

In this paper we showed the development of secured smart sensor platforms. Two prototypes are used to evaluate (1) the energy harvesting possibilities that arise when using NFC technology, and (2) the energy consumption of the main components of a smart sensor. These results are then used to implement simulations of the sensors. The simulations are able to not only simulate the smart sensor but also the environment. This is then used to answer crucial questions about the efficiency of the energy harvesting possibilities and the influence the environment has on the communication.

Acknowledgements

This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 692480. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Netherlands, Spain, Austria, Belgium, Slovakia.
IoSense is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2016 and May 2019. More information https://iktderzukunft.at/en/

REFERENCES

ams AG (2006). ams AS3955 NFC Interface Tag. http://ams.com/eng/Products/Wireless-Connectivity/Sensor-Tags-Interfaces/AS3955.

Chen, G., Fojtik, M., Kim, D., Fick, D., Park, J., Seok, M., Chen, M.-T., Foo, Z., Sylvester, D., and Blaauw, D. (2010). Millimeter-Scale Nearly Perpetual Sensor System with Stacked Battery and Solar Cells. In *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*. IEEE.

Choi, W. J., Jeon, Y., Jeong, J.-H., Sood, R., and Kim, S. G. (2006). Energy harvesting MEMS device based on thin film piezoelectric cantilevers. *Journal of Electroceramics*, 17(2-4):543–548.

Dziurdzia, P. and Stepien, J. (2011). Autonomous wireless link powered with harvested heat energy. In *2011 IEEE International Conference on Microwaves, Communications, Antennas and Electronic Systems (COMCAS 2011)*. IEEE.

Gummeson, J., Priyantha, B., and Liu, J. (2014). An Energy Harvesting Wearable Ring Platform for Gesture Input on Surfaces. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, pages 162–175, New York, NY, USA. ACM.

Halgamuge, M. N., Zukerman, M., Ramamohanarao, K., and Vu, H. L. (2009). An Estimation of Sensor Energy Consumption. *Progress in Electromagnetics Research*, 12:259–295.

Kansal, A., Hsu, J., Zahedi, S., and Srivastava, M. B. (2007). Power Management in Energy Harvesting Sensor Networks. *ACM Trans. Embed. Comput. Syst.*, 6(4).

Kansal, A. and Srivastava, M. (2003). An Environmental Energy Harvesting Framework for Sensor Networks. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design, 2003. ISLPED 2003.* ACM.

Kim, H. S., Abdelzaher, T. F., and Kwon, W. H. (2003). Minimum-Energy Asynchronous Dissemination to Mobile Sinks in Wireless Sensor Networks. In *Proceedings of the first international conference on Embedded networked sensor systems - SenSys 2003*. ACM Press.

Marta, M. and Cardei, M. (2009). Improved sensor network lifetime with multiple mobile sinks. *Pervasive and Mobile Computing*, 5(5):542–555.

Pieber, T. W., Ulz, T., and Steger, C. (2017a). SystemC Test Case Generation with the Gazebo Simulator. In *Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH,*, pages 65–72. INSTICC, SciTePress.

Pieber, T. W., Ulz, T., and Steger, C. (2017b). Using Gazebo to Generate Use Case Based Stimuli for SystemC. In *International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pages 241–256. Springer.

Pieber, T. W., Ulz, T., Steger, C., and Matischek, R. (2017c). Hardware Secured, Password-based Authentication for Smart Sensors for the Industrial Internet of Things. In *International Conference on Network and System Security*, pages 632–642. Springer.

Pinuela, M., Mitcheson, P. D., and Lucyszyn, S. (2013). Ambient RF energy harvesting in urban and semi-urban environments. *IEEE Transactions on Microwave Theory and Techniques*, 61(7):2715–2726.

Rahimi, M., Shah, H., Sukhatme, G., Heideman, J., and Estrin, D. (2003). Studying the Feasibility of Energy Harvesting in a Mobile Sensor Network. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. IEEE.

Shah, R. C., Roy, S., Jain, S., and Brunette, W. (2003). Data MULEs: Modeling and Analysis of a Three-tier Architecture for Sparse Sensor Networks. *Ad Hoc Networks*, 1(2-3):215–233.

Tan, Y. K. and Panda, S. K. (2011). Energy Harvesting From Hybrid Indoor Ambient Light and Thermal Energy Sources for Enhanced Performance of Wireless Sensor Nodes. *IEEE Transactions on Industrial Electronics*, 58(9):4424–4435.

TI (2014). MSP430FR5969 LaunchPad (TM) Development Kit User's Guide. http://www.ti.com/lit/ug/slau535b/slau535b.pdf.

Ulz, T., Haas, S., Pieber, T., Steger, C., and Matischek, R. (2017a). Sneakernet on Wheels: Trustworthy NFC-based Robot to Machine Communication. In *2017 IEEE International Conference on RFID Technology & Application (RFID-TA)*, pages 260–265. IEEE.

Ulz, T., Pieber, T., Steger, C., Matischek, R., and Bock, H. (2017b). Towards trustworthy data in networked control systems: A hardware-based approach. In *Emerging Technologies and Factory Automation (ETFA), 2017 22nd IEEE International Conference on*, pages 1–8. IEEE.

Yan, R., Sun, H., and Qian, Y. (2013). Energy-aware sensor node design with its application in wireless sensor networks. *IEEE Transactions on Instrumentation and Measurement*, 62(5):1183–1191.

Ye, F., Luo, H., Cheng, J., Lu, S., and Zhang, L. (2002). A Two-Tier Data Dissemination Model for Large-scale Wireless Sensor Networks. In *Proceedings of the 8th annual international conference on Mobile computing and networking - MobiCom 2002*. ACM Press.

Zhang, F., Zhang, T., Yang, X., Zhang, L., Leng, K., Huang, Y., and Chen, Y. (2013). A high-performance supercapacitor-battery hybrid energy storage device based on graphene-enhanced electrode materials with ultrahigh energy density. *Energy & Environmental Science*, 6(5):1623.

Simulating a Network:

An Approach for Connecting Multiple SystemC Simulations

Thomas W. Pieber, Fikret Basic, Thomas Ulz, and Christian Steger
Institute for Technical Informatics
Graz University of Technology
Graz, Austria
Email: {thomas.pieber, basic, thomas.ulz, steger}@tugraz.at

***Abstract*—Nowadays, sensor networks are widely used. To create new hardware for these networks, simulations are used. These simulations help during the design of the sensor nodes by providing information about internal states, power usage, and expected lifetime. They are useful to design one piece of hardware, however they are cumbersome when multiple of such hardware simulation instances need to interact with each other. This paper explores the possibility of starting multiple instances of a hardware simulation and connecting these via a simulation of the environment they will be used in.**

Keywords–SystemC; Simulation; Parallel; Network.

I. Introduction

The concurrent development of hardware and software enables the accurate simulation of the behavior of a finished sensor node. Rather than focusing solely on the hardware simulation, languages such as SystemC can also calculate the software influences on the system. This is a necessary step towards the complete simulation of a network. To simulate the behavior of a network, multiple such sensor instances need to be simulated simultaneously. These virtual sensor nodes are then connected via a suitable environment simulation.

There are many solutions to the problem of simulating single pieces of hardware. Hardware Description Languages (HDLs) support the simulation of hardware on a low layer of abstraction. The hardware descriptions used in this kind of simulation typically enable the manufacturing of the hardware itself. This means that the description, and thus the simulation, of the device is very accurate. This accuracy comes with the cost of low performance. To simulate more complex systems, a higher layer of abstraction is needed. This increases the simulation speed but decreases the resulting accuracy. A HDL that supports multiple layers of abstraction, such as SystemC [1], can describe the interconnection of hardware components at a high layer of abstraction and, at the same time, keep most of the accuracy for the components themselves [2].

The simulation of a network of computing devices can be challenging. Most of the simulators available are either limited to the sole simulation of the interaction between the network components (e.g., [3]), or can just give rough estimations of the executing time on each component (e.g., [4]).

To be able to simulate a sensor network without sacrificing the accuracy requires the connection of a hardware simulation tool and some tool that performs the interaction between the sensor nodes. Pieber et al. [5][6] described an approach to connect one instance of a SystemC simulation to the Gazebo simulator, a tool that is able to simulate an environment for the sensor. We want to extend this approach by instantiating multiple sensors in the Gazebo simulation and connecting them in this environment.

To test the resulting performance change, a small-scale networked control system has been implemented. This system is then performing some data acquisition, forwarding the data through another network node and finally the data storage at a final sensor node. In contrast to the simulation created by Pieber et al. [5] where one SystemC instance is connected to the Gazebo simulation, this simulation connects multiple instances of SystemC simulations to the Gazebo environment.

The remainder of this paper describes this process. Section II briefly describes the background information and states the related work to this publication. In Section III, the design of our approach is described. The implementation of the necessary parts is described in Section IV. Section V highlights our findings. This publication concludes in Section VI. This section also states our thoughts of what can be done next.

II. Background and Related Work

As concurrent tasks are a vital part of any simulation, mechanisms to cope with this are introduced in every modelling language. There are three main approaches to deal with concurrency in simulations:

1) **Multi-instance-one-simulation** One simulation contains multiple instances of parallel tasks.
2) **One-instance-multi-simulation** One model only contains a single task. These tasks are then run in multiple simulations that communicate with each other.
3) **Multi-instance-multi-simulation** Each simulation contains multiple concurrent tasks. The tasks communicate within a simulation, the simulations communicate within themselves.

A simulation written in SystemC and Gazebo are simulations of the first type. SystemC simulations run its component modules quasi parallel using delta-cycles to reach a stable state. Then, the simulation time is advanced until the next triggered event happens. In the Gazebo simulator, the components are modeled via plugins. These plugins are called sequentially. When all components have been executed, the simulation time is advanced by one time increment and the process restarts.

A more powerful example of a SystemC simulation comprising of multiple instances of modules has been created by

Original work published in Proceedings ...

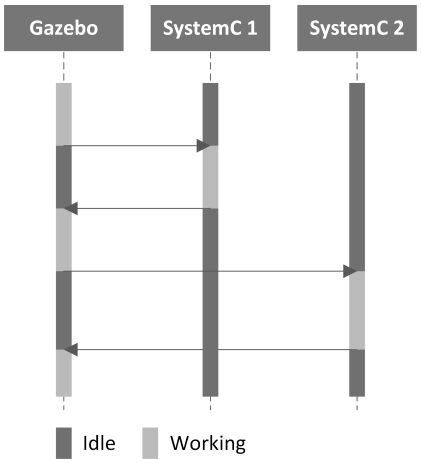

Figure 1. Performance problem using Gazebo and multiple SystemC instances.

Park et al. [7]. In this simulation, a smart house was simulated using SystemC. There, each electrical component has been modeled as a part of the complete simulation. These models are evaluated in parallel.

The combination of these two simulations results in a concurrent simulation of the third type. This is a simulation with multiple instances of different simulations, each containing multiple concurrent modules.

The process Pieber et al. [6] described in their work is intended to simulate one sensor node in an environment. This environment provides the sensor with stimuli that the sensor can work with. Furthermore, a communication channel is provided to communicate with the sensor and get feedback during the simulation time. In this approach, the Gazebo simulation is halted during the execution of one SystemC simulation. This results in a performance issue when using multiple SystemC instances. Figure 1 illustrates this. Here, one SystemC simulation blocks all other simulations. This results in an execution pattern suitable for a single processor core where only one simulation is being executed at a time. In this paper, we try to parallelize the SystemC simulations by introducing a mechanism that starts all SystemC simulation steps in parallel and waits for all to finish.

There are some proposed solutions to parallelize SystemC, such as [8]-[11]. These describe approaches that look for all executable SystemC modules and try to execute them in parallel. This results in a Type 2 concurrent simulation (One-instance-multi-simulation) as each executable model is treated as a single simulation. In these approaches, one large simulation is split into multiple concurrent simulations that are spread over the cores of one computer. In contrast to this, the approach described here uses multiple SystemC simulations and distributes them via a network. This method can spread the simulation on cores of the same machine, but also use additional computational resources of other computers in the network.

Schumacher et al. [9] presented an approach to simulate an Multi-Processor System on Chip (MPSoC). As their solution uses threads to achieve the parallelism, the solution is tied to a single host machine. While the authors claimed a significant performance improvement, it is already argued that this approach is not sufficient for a large sensor network simulation.

The approach of Sinha et al. [11] splits one SystemC simulation into multiple executable processes. In this approach, not only multiple cores or the Central Processing Unit (CPU) of one computer can be utilized, but also the Graphics Processing Unit (GPU).

Chopard et al. [12] propose a method to parallelize the SystemC kernel. In their approach, the researchers achieve a speedup comparable to the number of usable CPU cores.

The article of Jones [13] describes a more optimistic approach of parallelizing SystemC simulations. Jones also addresses the topic of race conditions that can occur when running such simulations in parallel. He also mentions that his technique of accelerating SystemC simulations is not suitable for existing simulations as large portions of code would need to be modified.

The possibility to accelerate SystemC simulations that rely on discrete events is discussed by Dömer et al. [14]. This team of researchers present a scheduler that spreads the runnable simulation nodes on the available CPUs.

Huang et al. [15] presented a SystemC library to handle the distribution of SystemC simulations. This approach is suited to work for multi-core machines as well as for a number of separate hosts. A downside of this approach is the limitation to functional and Transaction Level Modelling (TLM) simulations. Another disadvantage of this approach is the need for every SystemC simulation to handle its own communication with the rest of the simulation. Both of these issues are reflected on in our work by enabling SystemC simulations to be independent of the distribution architecture.

Another approach for simulating multiple computers in a network is used by Simics [3]. This simulator is built such that it can use multiple computers in a network to simulate the interaction of the systems. In this approach, the simulated network nodes are connected via a simulated network. This simulation can be used to simulate a computer network at a high level of abstraction. The integration of analogue signals (measurements of a sensor, or a very low level of abstraction) is not directly possible.

Clement et al. [16] coined the term Internet of Simulation. In their paper, the authors describe the need for heterogeneous simulation systems that can capture the complex nature of Cyber-Physical Systems (CPS). In their terminology, this paper describes a co-simulation for virtual engineering.

This paper is based on the publications of Pieber et al. [5][6]. It improves in the following details:

- **Concurrency:** The original approach uses one Gazebo plugin for each sensor in the environment. Each of these plugins directly creates a SystemC process and communicates with it. This entails that

all SystemC processes are located on the same host machine. With the approach presented here, the SystemC instances are started before the main simulation. These simulation instances can be located on different host machines and communicate via a network to the main communication. The plugins in the main simulation communicate to a server plugin that handles the network traffic and connects the SystemC simulations to the intended plugins.

- **Modularity:** As the SystemC simulations are started before the main simulation, multiple different implementations of the same simulation can be connected to the same sensor plugin. This increases the modularity of the simulation as only few changes need to be made in order to exchange the SystemC simulations.

III. Design

To improve the design of Pieber et al. [6] we implemented a server-client structure to spread the simulations to multiple computers. Using this, only a single plugin (the server) connects the SystemC instances. This plugin then blocks the Gazebo simulation until all SystemC tasks are finished. Figure 2 shows the intended execution path. The Gazebo simulator performs the calculations that are necessary for the data transmission. The SystemC simulations receive the data and perform operations on the data. When the simulation steps of all SystemC instances are finished the Gazebo simulator can continue its operation.

In this design, all necessary data from Gazebo is generated during its time step. This information is gathered in the server plugin. The server plugin then forwards the information to the SystemC instances. While the SystemC simulations are performing the simulation step, the Gazebo simulation is halted. During the execution of the SystemC simulation, the generated data is transmitted to the Gazebo simulation. There the server plugin captures the data. When the simulation step is finished, the SystemC simulations are halted and the Gazebo simulation can continue. In this way, no information is lost between the simulations, and all simulations are synchronized at the end of the time steps. As the server starts all SystemC simulations and waits for all to finish, also the SystemC simulations are synchronized.

In this design, the server does not contain simulation relevant operations. It just connects the Gazebo representations of the sensors (sensor plugins) to the SystemC simulations. This server therefore acts as a gateway for the sensor plugins to the SystemC simulations which can be executed anywhere in the network.
The server plugin has three responsibilities:

- **Connecting** the correct SystemC simulation to the intended sensor plugin. This includes the identification of the connected SystemC simulations and the matching to the correct sensor plugin.
- **Passing data** between the sensor plugin and SystemC simulation.
- **Synchronizing** the simulations.

The server plugin runs a thread that listens for incoming connections from remote SystemC simulations. For each connecting simulation, a thread is created that handles the information exchange with the sensor plugin. Additionally, this thread forwards the information to the SystemC simulation and receives the resulting information. Figure 3 shows the top-level structure of the plugin server. The server listens for new SystemC connections and creates a worker thread for each connected simulation. Each worker thread manages the communication between the sensor plugin and the SystemC simulation. To handle the SystemC simulation and the synchronization of the simulations, the thread appends data about simulation states. In addition to all of that, the thread keeps information about the connection to the SystemC simulation. This is information about the Internet Protocol (IP)-address, the port number, the simulation identifier, and the last sent command to which the SystemC simulation has to react.

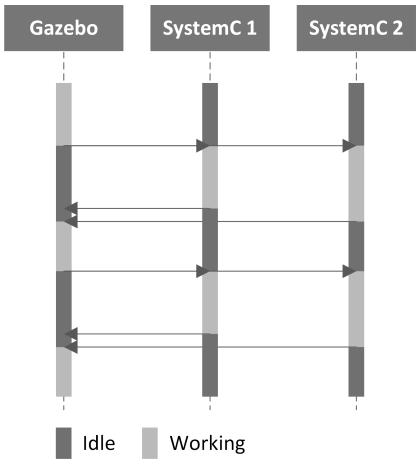

Figure 2. Improved handling scheme for multiple SystemC instances.

Each worker thread starts by initializing its own memory. This is followed by the initial checks of the SystemC simulation. These checks are performed by verifying an identifier. If the SystemC simulation can be used for the plugin, an initial configuration for the SystemC simulation is sent. When the initialization is finished, the SystemC simulation needs to respond with its state. For each simulation step, the data for the SystemC simulation is gathered by the sensor plugin. This data includes the change of sensor data, information about incoming messages, and changes of external energy sources (for energy harvesting). This gathered information is then forwarded to the server plugin and the correct worker thread. The worker thread packs the data and adds additional information. This additional information includes changes of simulation states, status information, or commands to the interface on the SystemC side. When the server plugin is being executed, all information is forwarded to the SystemC simulations by the worker threads. Until all worker threads have received the signal that their SystemC simulation has finished its execution, the Gazebo simulation is blocked. During the execution of the SystemC simulation, data that is destined for the Gazebo simulation is sent to the worker thread. This

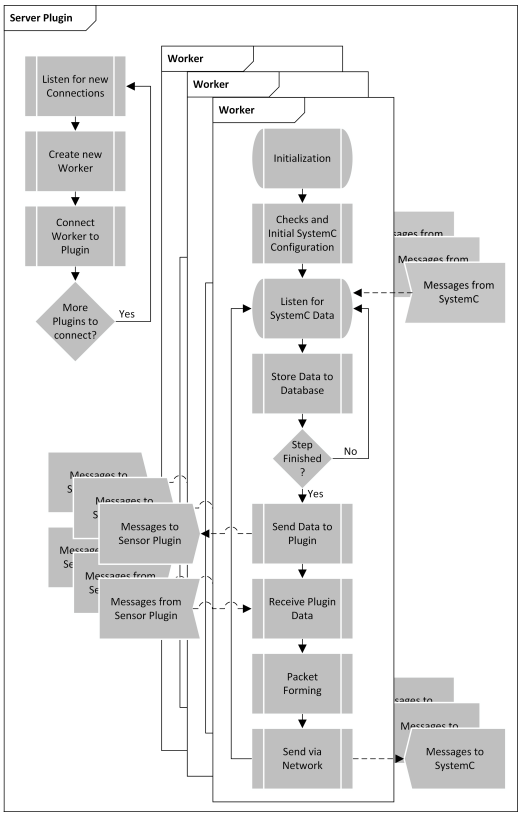

Figure 3. Top-level structure of the server plugin.

information is stored until the simulation proceeds. At the end of the SystemC simulation step, a signal is sent to the Gazebo simulation, informing the worker about the simulation status. If all SystemC simulations have stopped their execution, the Gazebo simulation can proceed. Now the stored data is transferred to the sensor plugins. The plugins can access the data in the next Gazebo time step.

With the use of the parallel design, the computation of the network should have a similar performance as the sole computation of the slowest node in the network. Thus, when simulating similar nodes, the performance gain should be related to the number of parallel nodes.

To test this hypothesis, multiple tests are designed:

1) **All nodes the same - single** This test simulates one sensor node performing measurements.
2) **All nodes the same - sequential** This test comprises of three identical nodes, each performing the same measurements.
3) **All nodes the same - parallel** The three nodes are run using the parallel computation design.
4) **Networked system - single** In this test each of three different nodes is simulated on its own. The data for nodes two and three is computed by node one and given as input for the simulation.
5) **Networked system - sequential** This tests simulates all three nodes in the network using the sequential approach. The data is generated and encrypted in node one, transmitted over node two, and decrypted and stored in node three.
6) **Networked system - parallel** The three nodes of the networked system are simulated using the parallel simulation design.

The first three tests act as a baseline test for the hypothesis that the simulation performance of the complete simulation is comparable to the simulation of the slowest node.

Test 1 simulates each node separately. As the three nodes are identical the simulation time should be equal as well. Test 2 uses the old connection via Gazebo to test the simulation speed of the sequential case. Each SystemC simulation is identical. As they are executed sequentially, the simulation time should be roughly the sum of the single simulations - in this time three times the duration of Test 1. Test 3 uses the new parallel design to connect the sensors. As all simulations are calculated in parallel, the execution time should be similar to the simulation of one node of Test 1.

The second three tests use three different nodes. One that gathers data, one that transmits the data to the last one, and one that receives and stores the data. The data is continuously transmitted and sent across the nodes. This way all nodes are active all the time (except for the first message). Otherwise, the test would limit the execution to the sequential case.

Test 4 uses precomputed data to test the functionality of each node individually. Each node is simulated separately. Test 5 combines the three nodes using the old connection via Gazebo to test the sequential case. Test 6 uses the new parallel design to connect the nodes.

The resulting simulation represents the data flow between the sensor nodes on the environment scale. Furthermore, as the sensor nodes themselves are simulated using SystemC, the data processing on each sensor node is being calculated.

IV. Implementation

The most notable change to the design of Pieber et al. [6] is that a new plugin has been implemented that performs the communication tasks. In their publication, every sensor plugin communicates with the corresponding SystemC instance. We have implemented another plugin that performs the communication tasks for all sensor plugins. A concept for our implementation can be seen in Figure 4. There, the *sensor plugins* send their data to the *plugin server*. This server handles the communication with all *SystemC* instances. When the SystemC simulations have returned data, the *plugin server* forwards the information to the *sensor plugins* in the next time step.

The communication between the server and the SystemC clients is based on the concept described by Pieber et al. [6]. An additional top-level structure is added to the EXtensible

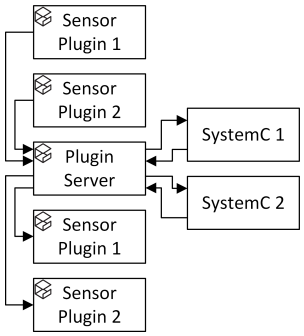

Figure 4. Concept for transmitting data between the sensor plugins, the plugin server, and the SystemC simulations.

Markup Language (XML)-formatted data. This structure specifies the type of message and the according payload. There are two different types of messages that can be sent between the server and client:

1) **Command:** This type specifies a message from the server that the simulation needs as input data. This can be a sensor value, incoming messages, or instructions to change the simulation status (finish the simulation, change the simulation step size).
2) **Status:** The status message can be sent from either side of the communication. If it is sent from the server, it can ask for the identification of the SystemC simulation, request information about the simulation status, or inform the SystemC simulation about its own simulation status. A status message from the client side can contain requested information, or signal the server that the simulation step is finished.

The complete communication structure between the server and a client can be seen in Figure 5. The Gazebo simulation starts the server. The server blocks the simulation until all SystemC clients are connected. After that, the server receives the commands to send from the sensor plugins. This information is relayed to the appropriate client. With this message the client is given the command to start the simulation step. Until all SystemC simulations have finished their step, the server blocks the Gazebo simulation. When all clients are ready, the Gazebo simulation can be resumed. This results in another message to the SystemC client. If the Gazebo simulation is to be ended, the server disconnects from the SystemC client. This triggers the reset of the SystemC simulation. As the simulation system is built to run in a network, simulation clients other than the intended ones could connect to the server. To enable the server to check if the SystemC simulation is required in the current Gazebo simulation run, an ID is requested from the client. This ID specifies the type of simulation. Based on this, the server can decide whether the client should be accepted or rejected. Accepted clients are then logically connected to the appropriate sensor plugin.

During the execution of the SystemC tasks, the plugin server blocks the Gazebo simulation. When all SystemC instances have returned their signal that the simulation step has been finished, the plugin server resumes by distributing the received information to the sensor plugins.

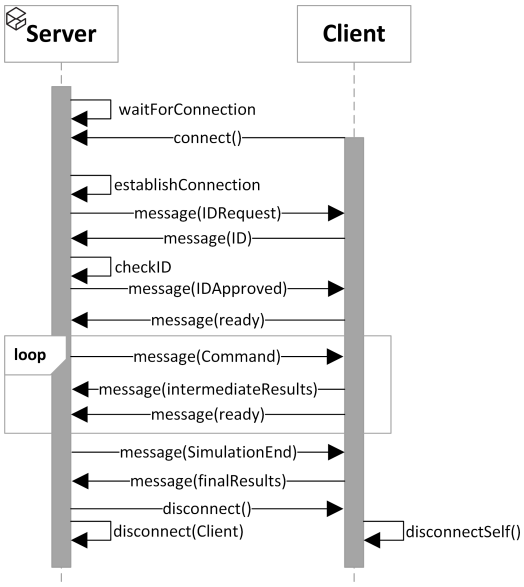

Figure 5. Messages between the server and one SystemC client.

TABLE I. Durations of the test simulations

Test Nr.	Node Nr.	Duration
1	1,2,3	$\sim$ 15.34 sec
2	1,2,3	48.0487 sec
3	1,2,3	16.7513 sec
4	1	153.7706 sec
4	2	10.6859 sec
4	3	120.8714 sec
5	1,2,3	284.6381 sec
6	1,2,3	156.1416 sec

V. Results

The results of the experiments introduced in Section III are listed in Table I.

These six experiments show that the final execution time is slightly larger than the longest component simulation. It furthermore shows that the simulation time, compared to the sequential case, can be multiplied with a factor of $\sim \frac{1}{N}$ where N is the number of parallel SystemC simulations where each simulation uses approximately the same amount of time. For simulations that differ in their simulation duration, the final duration is slightly longer than for the slowest simulation. This system therefore provides an extensible and flexible basis to add additional SystemC simulations. The Gazebo server remains independent of any added SystemC simulations while also the client side keeps individual SystemC models separated.

As an additional improvement can be seen that the sim-

ulations need not be calculated on the same computer as the Gazebo simulation. Normally, the Gazebo host system would need additional resources to run the Gazebo system and the SystemC simulation. The server-client structure allows the SystemC simulations to be spread over a network. Thus, the amount of possible parallel simulations is not limited to the resources on one machine.

As a limitation to this system, the general network overhead should be mentioned. As the commands and data are sent over a network, additional data is added by the system. This adds to the data size that is to be sent. Furthermore, the data needs to be packed and unpacked at either side of the communication, increasing the latency. This should be considered, when designing the simulation.

VI. Conclusion and Future Work

This paper describes a method to connect multiple SystemC simulations of sensor nodes to a network. These sensor nodes are placed in a virtual environment, simulated with the Gazebo simulator, and communicate via this environment with each other. As the Gazebo simulator processes its components sequentially, the final simulation is inefficient. To improve the performance of this simulation, a server-client structure is proposed and implemented. This connects the server on the Gazebo side to the SystemC simulations via network sockets. The server can then unite the individual calls to the SystemC simulations and start all simulations in parallel. In contrast to the simulation duration being the sum of all component simulations, the duration of the simulation using this structure is only slightly longer than the longest component simulation.

In the current form, the simulation results can only be evaluated when the simulation is finished. Due to the lengthy simulations, this is inefficient. Therefore, live signal plotting can be implemented. Using this, the simulation operator can spot errors in the simulation early and stop the execution prematurely. This would then reduce the simulation time for erroneous runs significantly.

Acknowledgments

This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 692480. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Netherlands, Spain, Austria, Belgium, Slovakia.

IoSense is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2016 and April 2019. More information https://iktderzukunft.at/en/

References

[1] Accelera, "SystemC." http://accellera.org/downloads/standards/systemc, 2000. Last accessed on mar 18, 2019.

[2] P. R. Panda, "SystemC - A modelling platform supporting multiple design abstractions," in *Proceedings of the 14th international symposium on Systems synthesis - ISSS*, pp. 75–80, Association for Computing Machinery (ACM), 2001.

[3] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[4] V. Shnayder, M. Hempstead, B.-r. Chen, G. W. Allen, and M. Welsh, "Simulating the Power Consumption of Large-scale Sensor Network Applications," in *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, (New York, NY, USA), pp. 188–200, ACM, 2004.

[5] T. W. Pieber, T. Ulz, and C. Steger, "SystemC Test Case Generation with the Gazebo Simulator," in *Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH,*, pp. 65–72, INSTICC, SciTePress, 2017.

[6] T. W. Pieber, T. Ulz, and C. Steger, "Using Gazebo to Generate Use Case Based Stimuli for SystemC," in *International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pp. 241–256, Springer, 2017.

[7] S. Park, H. Kim, H. Moon, J. Heo, and S. Yoon, "Concurrent Simulation Platform for Energy-Aware Smart Metering Systems," *IEEE transactions on Consumer Electronics*, vol. 56, no. 3, pp. 1918–1926, 2010.

[8] p. Ezudheen, P. Chandran, J. Chandra, B. P. Simon, D. Ravi, *et al.*, "Parallelizing SystemC Kernel for Fast Hardware Simulation on SMP Machines," in *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, pp. 80–87, IEEE Computer Society, 2009.

[9] C. Schumacher, R. Leupers, D. Petras, and A. Hoffmann, "parSC: synchronous parallel systemc simulation on multi-core host architectures," in *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pp. 241–246, IEEE, 2010.

[10] A. Mello, I. Maia, A. Greiner, and F. Pecheux, "Parallel Simulation of SystemC TLM 2.0 Compliant MPSoC on SMP Workstations," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 606–609, European Design and Automation Association, 2010.

[11] R. Sinha, A. Prakash, and H. D. Patel, "Parallel simulation of mixed-abstraction SystemC models on GPUs and multicore CPUs," in *17th Asia and South Pacific Design Automation Conference*, pp. 455–460, IEEE, Jan. 2012.

[12] B. Chopard, P. Combes, and J. Zory, "A Conservative Approach to SystemC Parallelization," in *International conference on computational science*, pp. 653–660, Springer, 2006.

[13] S. Jones, "Optimistic Pparallelisation of SystemC," *Universite Joseph Fourier: MoSiG DEMIPS, Tech. Rep*, 2011.

[14] R. Dömer, W. Chen, X. Han, and A. Gerstlauer, "Multi-Core Parallel Simulation of System-Level Description Languages," in *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pp. 311–316, IEEE Press, 2011.

[15] K. Huang, I. Bacivarov, F. Hugelshofer, and L. Thiele, "Scalably distributed SystemC simulation for embedded applications," in *2008 International Symposium on Industrial Embedded Systems*, pp. 271–274, IEEE, 2008.

[16] S. Clement, D. W. McKee, R. Romano, J. Xu, J. Lopez, and D. Battersby, "The Internet of Simulation: Enabling Agile Model Based Systems Engineering for Cyber-Physical Systems," in *2017 12th System of Systems Engineering Conference (SoSE)*, pp. 1–6, IEEE, 2017.