Kernel-based ARchitecture for safetY-critical cONtrol

KARYON

FP7-288195

D3.5 – Final Report on middleware and the evaluation environment

Work Package	WP3		
Due Date	37	Submission Date	2014-11-24
Main Author(s)	Jörg Kaiser (OVGU)		
Contributors	Christoph Steup (OVGU), Tino Brade (OVGU), Jeferson Souza (FCUL), José Rufino (FCUL), Rui Caldeira (FCUL), Daniel Skarin (SP), Benjamin Vedder (SP), Rolf Johansson (SP), Henrik Eriksson (SP), Christian Berger (CTHA), Oscar Morales (CTHA), Thomas Petig (CTHA), and Elad Michael Schiller (CTHA)		
Version	1.0	Status	Final
Dissemination Level	Public	Nature	Report
Keywords	Adaptive Middleware, Mixed Reality, Fault Injection, Reliable Communication		
Reviewers	António Casimiro (FCUL)		

SEVENTH FRAMEWORK PROGRAMME

Part of the Seventh

Framework Programme

Funded by the EC - DG INFSO

Version history

Rev	Date	Author	Comments
V0.1	2014-10-22	Christoph Steup (OVGU)	Preliminary Draft release
V0.2	2014-11-18	Tino Brade (OVGU)	Prefinal Release
V1.0	2014-11-24	António Casimiro (FCUL)	Final review and delivery

Glossary of Acronyms

ASIL	Automotive Safety Integrity Level
CBS	Cyber Physical System
COTS	Commercial-of-The-Shelf
DoW	Description of Work
FCS	Frame Check Sequence
KARYON	Kernel-based Architecture for safetY-critical cONtrol
LDM	Local Dynamic Map
MAC	Medium Access Control
WP	Workpackage
PHY	Physical Layer
PID	Proportional-Integral-Derivative
RF	Radio Frequency
ROS	Robot Operating System
V-REP	Virtual Robot Experimentation Platform
V2X	Vehicle to X (X: Vehicle, Infrastructure)

Executive Summary

D3.5 summarizes the final development of WP3: Supporting technologies. After having provided working software prototypes in D3.3, this report gives an overview over the adaptations and additions that have been made to the middleware with respect to the demonstrators and particularly the evaluation environment comprising fault injection. As indicated in the DoW, this report is mainly composed from technical papers. The document is structured in topics along the tasks in WP3. The chapters are devoted to a certain topic each. They contain short informal descriptions of the more detailed papers that are allocated in the Annexes.

Chapter 2 of the report is dealing with resilient real-time communication over wireless media. A novel architecture is described that is based on an abstract communication model, which offers a set of correctness, dependability and timeliness properties.

Chapter 3 describes the use of adaptive middleware in a focussed demonstrator. Here, we show the benefit when integrating hardware and software components in a mixed reality scenario that is based on quite a number of different technologies.

The various tools for fault injection are presented in Chapter 4. Fault injection particularly addressing communication faults is described in two papers. Section 4.1 presents a real-time assessment suite for IEEE 802.15.4 networks. This suite provides two services, namely network monitoring and fault-injection. With this suite effective fault injection campaigns can be easily defined and instantiated while providing means to visualize and record its effects. The work in Section 4.2 shows a way of using fault injection as part of a safety case for cooperative automotive systems. The failure modes are derived from respective safety standards and the prototype shows that it is feasible to inject all faults needed in a safety assessment according to the requirements in the functional safety standard for the automotive domain. Finally, Section 4.3 addresses a failure injection framework for the perception system needed for autonomous and cooperative driving. This is tightly related to the work on sensor failure modes and semantics performed in WP2. The white paper describes how patterns of sensor failures are defined and injected.

Chapter 5 includes final work on exploiting environment models for autonomous self-driving cars. Section 5.1 describes one of the key services implemented in the Gulliver Testbed: the Local Dynamic Map (LDM) and how it is exploited for "driving with confidence". An LDM, based on on-board and remote sensory information provides the position of all nearby noticeable objects along with the LDM's confidence about these positions. The paper describes how safety can be maintained even under the uncertainties of environment perception by changing the levels of service.

The paper summarized in Section 5.2 addresses the problem of how to store and retrieve environment information in a way that is relevant in the context of an application. The work classifies relevant types of environment data and how this data can be organized. The distributed Cassandra database is used, which allows every entity to store its data for its own purpose, but also to share it with interested entities if needed. A specifically tailored language "SelectScript" allows it to extract information supporting application-based environment models and views.

Table of Contents

List of Figures

List of Tables

1. Introduction

This final report on middleware and the evaluation environments covers the recent results in networking, the adaptive middleware and fault-injection. Monitoring and fault injection tools for dependable networking and adaptive middleware have been provided as software modules in D3.3. This final report describes additional components for developing and testing the middleware used in the demonstrators and final results in fault-injection.

The structure of the report follows the specification in the DoW and includes brief descriptions of new components added to the adaptive middleware and evaluation environments which are described in detail by technical papers attached in Annex A to Annex G.

The chapters are organized along the tasks of WP3. The next chapter comprises a contribution from T3.1 Predictability and Resilience in Embedded Networks. Chapter 3 addresses issues of "Adaptive Middleware for Advanced Control Systems". Chapter 4 is related to "Evaluation Environment and Tools" and Chapter 5 describe components to handle and exploit environment information.

2. The Wi-STARK Architecture For Resilient Real-Time Wireless Communications

This section presents an overview of the state of the art architecture dubbed Wi-STARK. The Wi-STARK architecture supports the provision of dependability and real-time guarantees within a one-hop communication domain established by the Medium Access Control (MAC) and Physical (PHY) communication (sub)layers between wireless nodes. Wi-STARK takes a divide-to-conquer approach to the provision of dependability and real-time guarantees, motivated by the following observation:

> If no real-time guarantees can be offered within communications at one-hop of distance, no real-time guarantees can be offered within multiple-hop communications at all.

That means, any dependable real-time message delivery guarantee has to be secured first within the one-hop of distance wireless space, prior to be extended end-to-end, across multiple hops. Given this context, the Wi-STARK architecture is deemed extremely useful and emerges as a fundamental building block to reach the ambitious goal of enforcing end-to-end real-time wireless communication guarantees, which have been the main target of several works in the literature [1] [2] [3].

The Wi-STARK architecture is presented in Figure 1. Its design is open and flexible, being composed by two layers dubbed *Channel Layer* and *Mediator Layer*, wrapping a standard service MAC sublayer and interfacing, at the bottom, a standard basic MAC and PHY layers (abstracted in Figure 1, by the RF transceiver interface). The design decision of wrapping the standard MAC sublayer has two main advantages: enables a tight control of the use of RF communication channels; allows to improve the services offered to high level protocol layers.

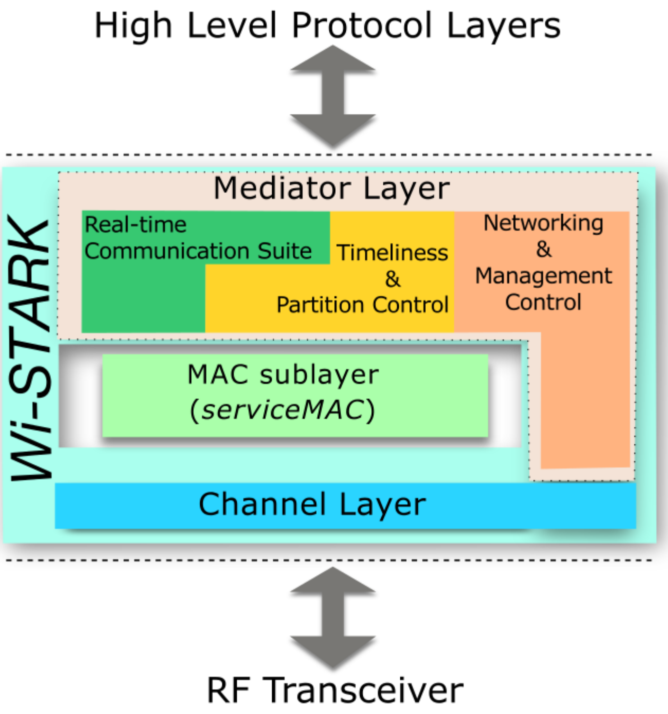

High Level Protocol Layers
Mediator Layer
Real-time Communication Suite
Timeliness & Partition Control
Networking & Management Control
Wi-STARK
MAC sublayer (*serviceMAC*)
Channel Layer
RF Transceiver

Figure 1 - The Wi-STARK architecture.

Detailing the architecture of Figure 1, the *Channel Layer* is a thin layer that provides a common interface to transparently control the use of a given RF communication channel for purposes of frame transmission and reception, incorporating useful extensions to enhance the dependability of communications. In particular, the *Channel Layer* implements: an extension to the standard Frame Check Sequence (FCS) mechanism (specified in [4]), which secures the detection and signalling of every error observed in received frames; the detection of RF communication channel failures; and the RF communication channel switch strategy, specified in [4], to be performed upon channel failure detection.

The MAC sublayer illustrated in Figure 1 is the standard MAC sublayer present in the traditional wireless networking protocol stack, such as those specified within the IEEE 802.15.4 [5] and IEEE 802.11p [6] wireless standards. In the context of the Wi-STARK architecture such standard MAC sublayer is dubbed *serviceMAC*, offering only conventional unreliable data frame and layer management service interfaces. No modifications to the standard specification are needed for its integration in the Wi-STARK architecture. In this sense, the Wi-STARK architecture is highly flexible supporting the integration of any MAC sublayer wireless protocol, including the real-time variants proposed in [1] [2].

The Mediator Layer is an extensible sublayer, specially designed to mediate the communication flow from (and to) the high level protocol layers, being established upon the exposed *serviceMAC* interface, as illustrated in Figure 1. The *Mediator Layer* is responsible for the semantically rich service interface offered by Wi-STARK, which includes reliable communications, effectively augmenting the services offered by the standard MAC sublayer.

In the perspective of networking communication protocol developers, the dependability and timeliness guarantees offered by the Wi-STARK architecture are represented by a set of fundamental primitives for transmission and reception of messages to/from the network, which are specified in Table 1.

Wi-STARK data service interface	
Primitives	**Description**
MLA.Data.request	Requests a message transmission using one of the Wi-STARK communication protocols.
MLA.Data.confirm	For reliable services, it confirms message delivery at recipients. Otherwise, it confirms only message transmission.
MLA.Data.indication	Notifies the arrival of a message.

Table 1 - Wi-STARK data service interface.

All of the primitives present in the Wi-STARK data service interface are easily integrated into embedded and real-time operating systems, being available as system calls associated to the wireless networking protocol stack. Design details of the Wi-STARK architecture can be found in the following contribution:

In Annex A: reprint of the paper "**The Wi-STARK Architecture For Resilient Real-Time Wireless Communications**". J. L. R. Souza and J. Rufino. EWiLi 2014. November 2014, Lisbon, Portugal.

3. The KARYON Adaptive Middleware

During the development of the automotive demo scenario of the Otto-von-Guericke University the KARYON Middleware was extended to incorporate more components. These additional components are mainly used to test, develop and visualize the behaviour of the demonstrator.

This section briefly describes the extension of the software as provided in D3.3 (Working Prototype of the Adaptive Middleware) and the use in the demonstrator.

3.1 The OvGU Demonstrator

The OvGU demonstrator aims to emphasize the abilities of integrating physical and virtual entities in a mixed-reality setup. Figure 2 depicts the example scenario. On the left side the real track and car are visible. The right side shows the virtual environment modelled in V-REP (Virtual Robotic Experimentation Platform) including the virtual representation of the real car in the lower part of the track as well as a virtual car in the middle of the track. The benefit of mixed reality is to allow and assist developers when gradually integrating functionality and test it at different levels of abstraction. Additionally, and in fact a consequence of mixed-reality, the demonstrator shows the fault-injection capabilities supported by the KARYON Middleware enabling fault analysis at varying levels of integration.

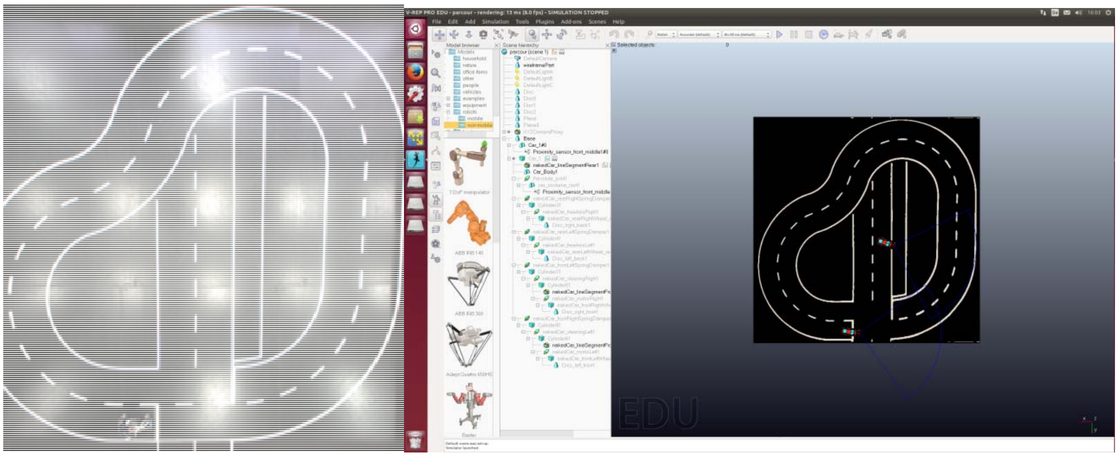


Figure 2 - Overview picture of the OvGU demonstrator.

3.1.1 Demo Scenario

The demonstration consists of three cars driving autonomously on the track. They follow the right lane of the track and keep their distance towards their front car. Two of them are virtual ones only driving in the simulation environment and one car is a real one driving in the simulation as well as on the real track. According to the KARYON architecture, the cars can drive at different levels of service depending on the quality of their perception of their position and orientation on the road. The quality of perception is in turn depended on the failure states of the various sensors expressed as validity of the respective sensor data. In the demonstration the level of service defines the maximum speed a car can travel. According to the validity of the sensor data the level of service and therefore the speed may decrease. In Annex B, we present the details of the interaction between speed, uncertainty and control of the car. Cars adapt their speed based on the local current level of service. In case of all sensors failing, the lowest level of service is activated, which switches to manual control (for practical reasons this is only available for the real car in the demonstrator).

It should be noted that all cars run the same control software. Mixed reality allows embedding the control software in three different system environments transparently:

1. We provide a real car that receives position data from a real camera-based sensor and controls real actuators. Both, the sensor data and the actuation commands are also fed to the simulator to enable mutual awareness between the real and the virtual cars. The real car uses the embedded hardware of the FCUL to provide a physically separated safety kernel controlling the level of service.
2. We run the control software as a hardware-in-the-loop system on the same embedded hardware that is used in the real car. However, the position information is completely derived from the simulation and the actuation commands are made available to the simulator. V-REP runs a complete simulation considering the physical properties of the car based on this information. Another safety kernel is run on the embedded hardware for this virtual car. Although, the physical separation of the safety kernel is not provided, it performs the same functionality in the demonstrator.
3. The control software runs on a powerful Linux machine receiving inputs and generating outputs as above.

It is quite obvious that a large number of different hardware and software components including the wireless links have to be integrated in the mixed-reality demonstration scenario. The KARYON Middleware is crucial to ease this integration task and provides a seamless interaction between all parts hiding the differences of the underlying system. Figure 3 roughly sketches the involved components. In this case, the position tracking sensors of the real car is visualized. To produce a position, a video stream is acquired through an industrial camera and fed to a ROS-based vision algorithm detecting the car in the video. Subsequently, the detected position is modified by a Matlab-based fault injection component, based on the current configuration. The configuration is supplied through the Android-User-Interface. To further highlight the benefits of the KARYON middleware, fault-injection exploits its specific publish-subscribe communication. Every sensor producing data for the demonstrator is enhanced with a Matlab-based fault injection component (as visible in Figure 3). These components are controlled via a specific fault-injection topic, which is accessed through a mobile tablet, to enable an interactive reconfiguration of the sensors' fault-injection parameters. This enables the visualization of the systems behaviour in different levels of service, without waiting for a specific fault. Finally the current state of the demonstrator, consisting of car positions and each car's level of service, is visualized on multiple displays. This allows the spectators to observe the systems internal behaviour.

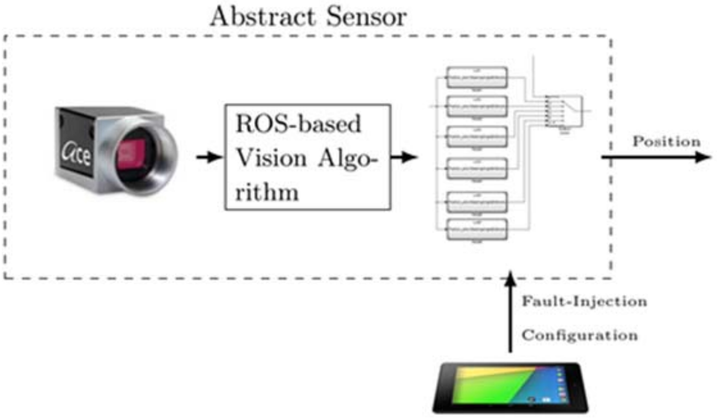


Figure 3 - A schematic view of an abstract sensor.

3.1.2 Extensions of the KARYON Middleware

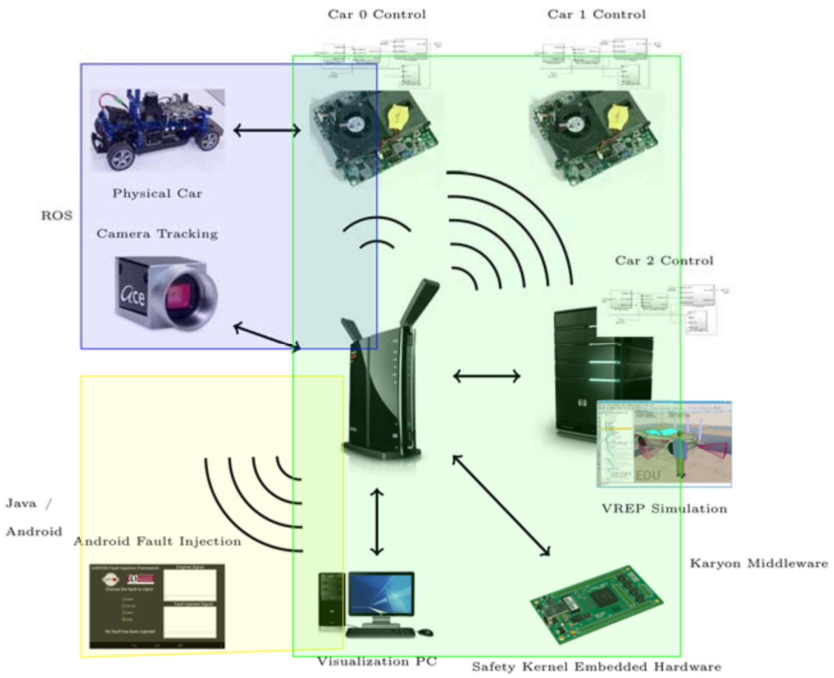


Figure 4 - Overview of the extended KARYON Middleware.

Figure 4 illustrates the extended KARYON Middleware used in the OvGU Automotive Demonstrator compared to the version that has been provided in D3.3. The blue parts of Figure 4 depict ROS-based Systems, yellow parts show Android/Java based systems and green parts use the native KARYON Middleware. The real car internally uses ROS-components (ROS: Robot Operating System) to handle hardware access and low-level computing tasks. The ROS-components accessing hardware are low-level drivers build for special devices like Hokuyo-laser-scanners or DC-motors. Additionally, specific functionalities like optical lane detection are provided by ROS. All of them, define their own message formats and message subjects that are used to communicate with other parts of the system using gateway nodes. These nodes transform the specific ROS-messages into publish-subscribe messages of the KARYON middleware enabling real-time properties as well as fault injection.

The mixed-reality of the simulation is achieved through the usage of VREP as a simulation environment. In this environment, virtual and real cars may coexist, which is only possible by synchronizing all cars' perception and actuation. The sensing and actuation of the virtual cars are handled directly by the simulation environment. The cars only need to fetch the sensor data from the simulation and transmit their actuation commands back. However, the real car is more difficult to include. On the one hand, the sensor data of the real car needs to be extended by the data inferred in the simulation. To this end, a virtual twin of the real car is included in the simulation, which is provided with simulated sensor data by the simulation environment. This simulated sensor data is fused with the real sensor data to form a consistent perception for the real car. On the other hand, the effects of the actuation of the real car need to be observed by the simulation environment to update the state of the virtual twin. Therefore, an optical tracking system observes the behaviour of the real car and publishes its information to the environment simulator. The optical tracking system uses ROS-based computer vision algorithms to derive the position and orientation of the real car. The Middleware allows, through the usage of publish/subscribe, a unification of the virtual sensors and actors and the real sensors and actors. Consequently, the control algorithms used in the cars perceive no difference between the data of a real or virtual entity enabling a transparent integration of real and virtual components.

The control software for the cars is implemented in Simulink, as depicted in Figure 5. The yellow blocks establish the communication link to the VRep environment simulator. Apart from providing basic communication functionalities, the special KARYON-Matlab-interface (Block A) synchronize the internal simulation time of Simulink to the synchronous time of the KARYON Middleware in order to achieve a consistent behaviour of the demonstrator. Afterwards, the environment model depicted by the green block (Block B) interprets the obtained sensor data, provides a model of the environment and generates an application specific view out of the environment model. Finally, the control software (Block C) operates on the generated view of the environment model and outputs respective actuator command that are fed back to the environment simulator via the KARYON-Matlab-interface (Block D). The mentioned structure is used for each car no matter whether simulation, hardware-in-the-loop or real hardware is used. In order to maintain such a perception-action loop, the KARYON-Matlab-interface keeps track of the necessary underlying primitives.

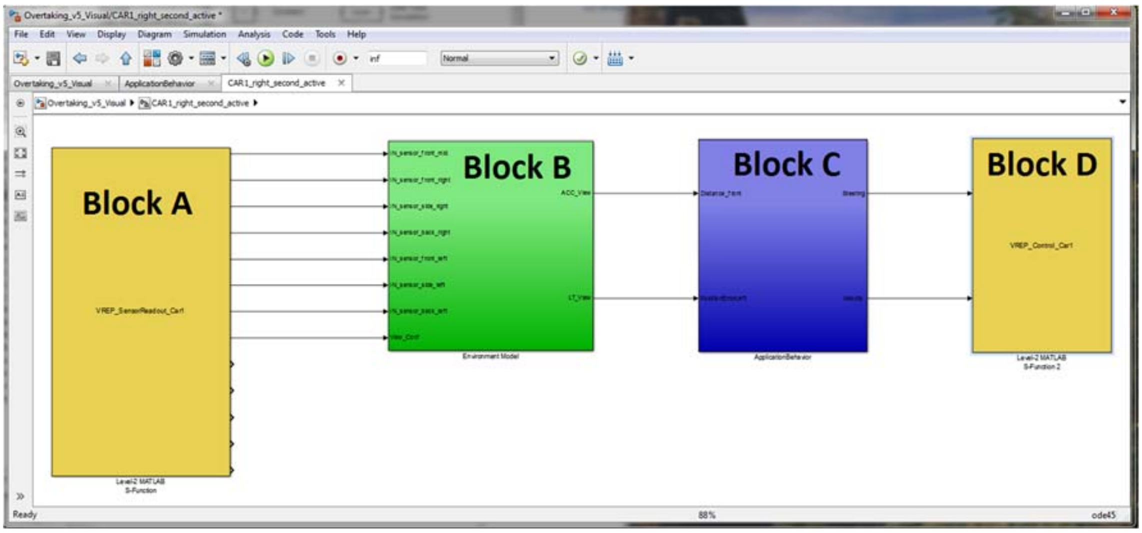


Figure 5 - Control behaviour modelled in Simulink.

Figure 6 provides an insight into the environment model consisting of the interpreter (Block E), the environment model (Block F) itself and the view generator (Block G), as described in D 2.6. For each running application an associated view is generated so that actual error values are provided based on which a respective application responds with an actuator command. This means the lane tracer and the adaptive cruise control are provided with relative position errors and relative distance values, respectively. Due to the achievements of the environment model, the application behaviour simplifies to a simple control loop as represented by the PID block illustrated in Figure 7. Dependent on the actual error value, the PID controller is used to keep the car on track and to maintain a gap between cars by respective steering and speeding commands.

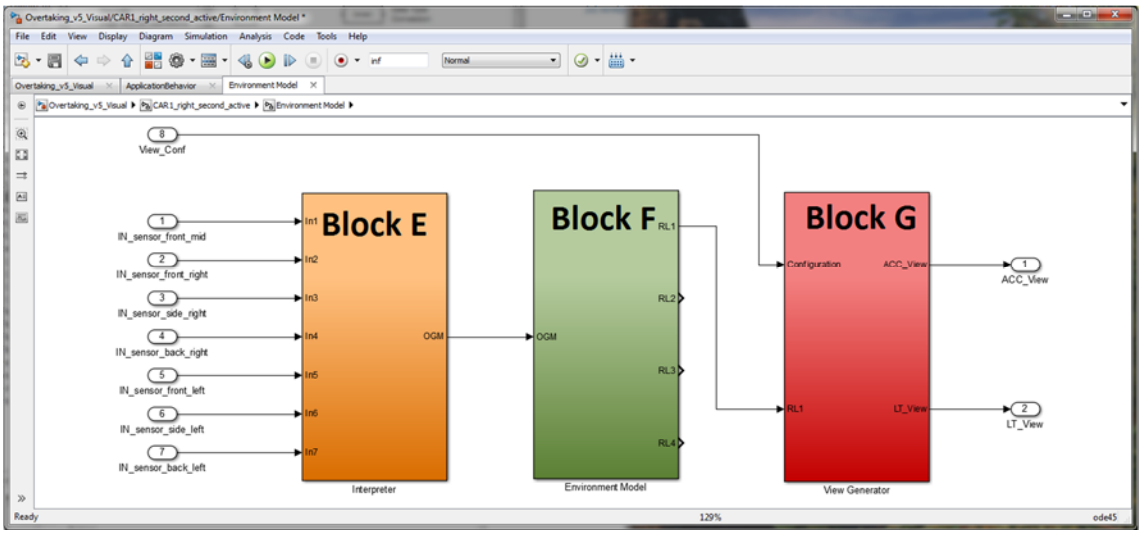
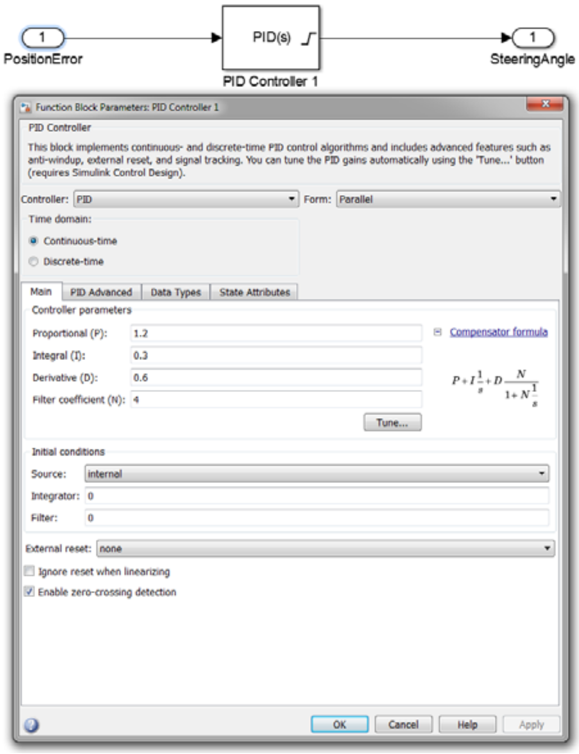


Figure 6 - The environment model with the interpreter and the view generator.



Function Block Parameters: PID Controller 1

PID Controller

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PID Form: Parallel

Time domain:

● Continuous-time

○ Discrete-time

Main | PID Advanced | Data Types | State Attributes

Controller parameters

Proportional (P): 1.2

Integral (I): 0.3

Derivative (D): 0.6

Filter coefficient (N): 4

Compensator formula

$$P + I\frac{1}{s} + D\frac{N}{1+N\frac{1}{s}}$$

Initial conditions

Source: internal

Integrator: 0

Filter: 0

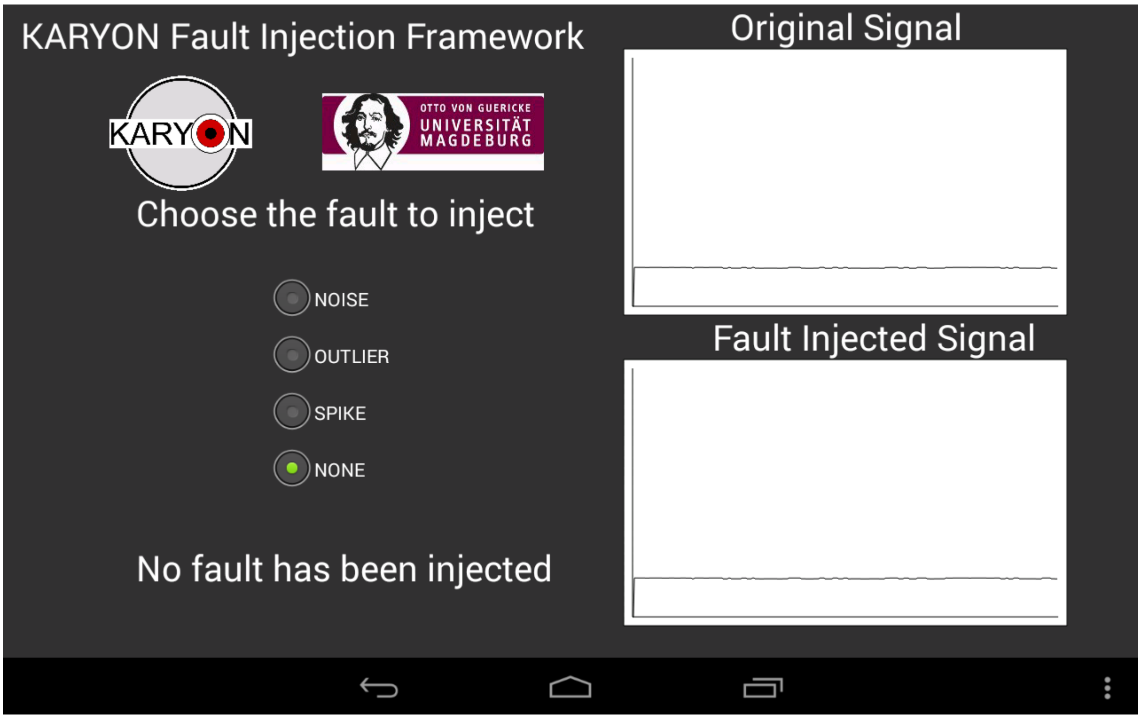
External reset: none

□ Ignore reset when linearizing

☑ Enable zero-crossing detection

Figure 7 - PID controller representing the application behaviour.

The visualization of the demonstrator's behaviour is done through the User Interface, which consists of two sub parts: A PC is used to visualize the current state of the mixed reality application. An Android Tablet is used to configure the fault injection framework of the KARYON middleware to different fault profiles, as shown in Figure 8. It uses the Java binding of the KARYON Middleware for integration.



KARYON Fault Injection Framework

Choose the fault to inject

○ NOISE
○ OUTLIER
○ SPIKE
● NONE

No fault has been injected

Figure 8 - The fault injection interface running on an Android tablet.

In Annex B : reprint of the paper "**Sensor-and Environment Dependent Performance Adaptation for Maintaining Safety Requirements**". T. Brade, G. Jäger, S. Zug and J. Kaiser. *Computer Safety, Reliability, and Security*, Springer, 2014, pp. 46-54.

4. Fault injection

4.1 Real-Time Assessment Through Fault Injection in IEEE 802.15.4 networks

Despite some advances in the support of real-time communication in wireless networks, there are still open problems to be solved, which often need the appropriate tools for their study and analysis, namely when experimental activities in real settings are of fundamental importance.

Obtaining experimental data from real network operation requires the use of appropriate tools immersed in the wireless network setting. Network monitoring is a technique aimed at analysing the functional network interactions between its nodes. Obtaining other non-functional characteristics such as reliability, timeliness, resilience and susceptibility to errors requires more advanced network monitoring tools. A tool aiming to extract both functional and non-functional characteristics from real wireless network settings will be, by itself, of extreme importance and significance for the study and analysis of network protocols.

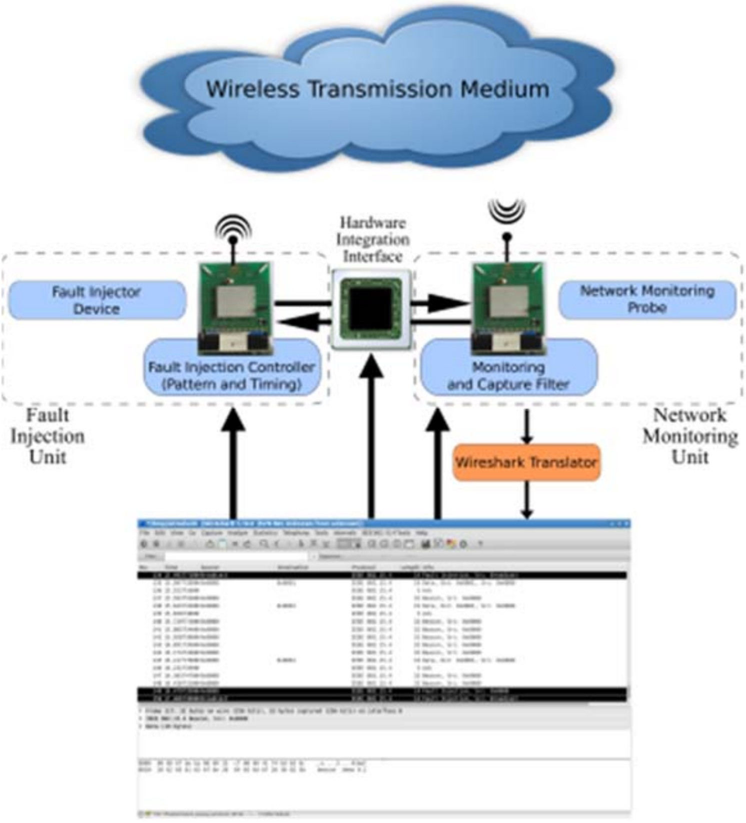

Wireless Transmission Medium
Hardware Integration Interface
Fault Injector Device
Fault Injection Controller (Pattern and Timing)
Network Monitoring Probe
Monitoring and Capture Filter
Fault Injection Unit
Network Monitoring Unit
Wireshark Translator

Figure 9 - System Architecture.

In addition, a key issue is that wireless communications are prone to be disturbed by electromagnetic interferences from the surrounding environment, causing then network errors. Since particular error patterns may be especially relevant for the analysis of the network operation and their natural occurrence may be rare, there is the need to re-enact such error patterns, trough fault injection. To fully understand how network errors interfere in the real-time behaviour of networking communications a real-time assessment toolbox is required. Both fault injection and networking monitoring functions are needed: the fault injector (selectively) enforces the occurrence of (particular) network errors, possibly in some kind of transmitted frame; monitoring and capturing the network traffic enables the analysis of the real-time capabilities of the communication system face to the injected error pattern. Some error patterns may be so relevant that they may be configured as pre-defined fault injection profiles.

Within the scope of the KARYON Project, a toolbox has been developed to assess not only the functional properties of wireless network operation but also a set of non-functional properties of network and protocols, such as reliability, timeliness, resilience and susceptibility to network errors. The toolbox architecture and design is completely general and the basic concepts and principles can be applied to any wireless network. The toolbox is composed by four components (as illustrated in Figure 9): the network monitoring unit, the fault injector unit, the hardware integration interface, and the Wireshark. A use-case has been set and engineered for the real-time assessment of IEEE 802.15.4 [5] networks and protocols.

The Network Monitoring Unit includes a Commercial-Of-The-Shelf (COTS) hardware network interface device acting as networking monitoring probe (a.k.a. sniffer) with the purpose of capturing all traffic transmitted through a specific channel. Taking advantage of the promiscuous mode to receive all correct traffic in conjunction with the Frame Check Sequence (FCS) extension specified in [4] the Network Monitoring Unit is able to receive all correct traffic as well as frame received with errors.

The Fault Injector Unit includes a fault injector device intended to force the occurrence of errors accordingly with a given (configurable) pattern and timing. Faults are injected directly in the wireless communication medium. In all the aspects, the Fault Injector Unit is a perfectly common Commercial-Of-The-Shelf (COTS) network interface, with the exception that the network interface may be configured to bypass the medium access control protocol thus allowing a direct access to the wireless transmission medium when needed. The Fault Injection Unit operation can be individually configured for a specific fault injection campaign or it can be configured with one of several pre-defined fault injection profiles.

A third unit, the Hardware Integration Interface connects the two previous units: the Network Monitoring Unit and the Fault Injector Unit. The Hardware Integration Interface is a glue component that enables extremely fast cooperation and interaction between the network monitoring unit and the fault injection unit. For example, a given fault injection event or campaign can be timely triggered by the detection of a given event or operational condition by the Network Monitoring Unit.

Finally, Wireshark [7] was extended to manage all aspects of the network assessment, including control and data collection from the previously mentioned components. Wireshark is a referenced network protocol analyser tool with support for different wireless network standards, including EEE 802.15.4 packet visualisation capabilities.

Further details on the tool above presented can be found in the following contribution:

In Annex C: reprint of the paper **“A Tool for Real-Time Assessment Through Fault Injection in IEEE 802.15.4 Networks”.** R. P. Caldeira, J. L. R. Souza and J. Rufino. Submitted for publication.

4.2 A Fault-Injection Prototype for Safety Assessment of V2X Communication

One of the key characteristic of a cooperative system is the possibility to wirelessly exchange information, usually local sensor data which is distributed globally. One common example is to exchange absolute and relative positions among the involved vehicles. Thus, the wireless communication links will be associated with safety requirement and their corresponding safety integrity levels. The integrity of the communication links will be monitored by safety mangers on-board the vehicles and the level-of-service will be adjusted accordingly to maintain a safe system. In order to evaluate the aforementioned functionality, it is necessary to have the possibility to inject faults in the communication links in a controlled and predictable manner. The fault-injection prototype presented in the paper of Annex D have been developed to fulfil this purpose.

The techniques used in the prototype are generic, i.e. agnostic with respect to underlying communication protocol. However, the presented implementation of the prototype targets the IEEE802.15.4 protocol used in e.g. wireless sensor networks. The prototype emulates many most common communication failure modes by one or a combination of packet sniffing, jamming, and insertion. As an example a message delay is emulated by jamming the original message and resending (injecting) it a specified delay later (a priori knowledge of message content is assumed). State machines are used to control fault injection triggers and the fault injection itself. The following failure modes are currently supported by the fault injection prototype: message corruption, delay, loss, insertion, unintended message repetition, masquerading, and blocking access.

In Annex D: reprint of the paper "**A Fault-Injection Prototype for Safety Assessment of V2X Communication**". Daniel Skarin, Benjamin Vedder, Rolf Johansson, and Henrik Eriksson, Department of Electronics, SP Technical Research Institute of Sweden. Presented at DEPEND 2014: The Seventh International Conference on Dependability. Published in conference proceedings by IARIA, 2014. ISBN: 978-1-61208-378-0.

4.3 Configuring Fault injection for Cyber-Physical Systems

Autonomous vehicles intimately link the entities of the real-world to their internal virtual world images. At the interface between these worlds, sensors perceive and transform the real-world entities. The challenge is to provide a precise and an accurate observation that may be substantially affected by sensor failures and inherent uncertainties. Obviously, these failures cause a potential risk for violating safety goals, because an application takes action based on these observations. A safety critical system must be proven safe before it will be put to operation due to the potential to cause harm. Therefore, checking, analysis and validation have to be performed at early design and development stages. Failure injection is one of the important techniques providing evidence at design-time that the application is either robust enough to resist failures or not susceptible to the considered set of failures. Recognizing the advantages, failure injection became part of several standard such as ISO 26262.

ISO 26262 defines automotive safety integrity levels (ASIL) that identify the safety requirements for a function that may be broken down to requirements of components that are involved. In order to verify correct operation, test patterns are demanded to be injected. The characteristics of such tests as coverage an extension depend on the safety integrity attribute (ISO 26262 - ASIL A-D, ISO 61508 - SIL 1-4). For instance, out-of-range, offsets, stuck in range and oscillations faults are specified by the standard. However, the standard the standard does not address complex fault patters of a sensor and the selection of the injection points (e.g. hardware or software) as well as the parameterization of the injected faults is left open.

The proposed failure injection framework provides means to model and inject complex failures. It allows to define the shape of a failure signal, its distribution over time and combinations of various failures, thus complex failure patterns. Because the failure injection is to some extent stochastic, it must be ensured that all failure patterns are injected with a sufficient occurrence rate. A failure injection monitor allows to observe the failures and to verify occurrence rates.

Sensor failures are specified by electronic data sheets. These descriptions are automatically transformed to Simulink blocks and placed between the correct sensor (whether a simulated sensor or a real sensor operated under lab conditions) and the respective control component. The KARYON middleware supports this configuration procedure (see also section 3.1.2). External configuration tools enable the injection of different complex failure patterns at run time, thus exercising the system under varying failure conditions.

In Annex E: reprint of the paper "**A Failure Injection Framework for Cyber-Physical Systems**". Sebastian Zug, Tino Brade, Christoph Steup. Submitted for publication. White Paper EOS-OVGU.

5. Exploiting Environment Models

5.1 Driving with Confidence: Local Dynamic Maps That Provide LoS for the Gulliver Test-Bed

The design of automated driving systems aims at reducing the human error and increasing the fuel efficiency by letting the vehicles map their surroundings and drive autonomously. One of the system challenges on the road is that at any time the environment can stop meeting the system's operational conditions (and then resume meeting the requirements at some later point in time). Thus, as vehicles map their surroundings, they should also provide information that can help the vehicles to know whether the operational conditions are met with respect to the confidence that they have about the mapped information.

We design and implement key services of Local Dynamic Maps (LDMs) that are based on on-board and remote sensory information. The LDM provides the position of all nearby noticeable objects along with the LDM's confidence about these positions. The design also includes an extension that allows the vehicular system to agree on the lowest common ability to meet the operational conditions.

We evaluate the performance of a key component in our pilot implementation together with a set of test cases that validate the proposed design. Our current findings show that the presented ideas can accelerate the deployment of automated driving systems.

Self-driving cars will be the next big step in vehicular technology as several important automotive original equipment manufacturers (OEMs) have recently announced. However, their specific challenge besides deploying a robust and reliable technology throughout a vehicle's lifetime is to bring down the technology's costs. Therefore, expensive sensors that perceive a vehicle's surroundings need to be substituted by cheaper counterparts. Cheap sensors normally have a reduced accuracy. This is addressed by sensor fusion with information provided by other vehicles and the infrastructure.

Research in this area however is time-consuming, error-prone, expensive, and tedious, when several cars need to be coordinated within a real-scale experiment on a real proving ground. As an intermediate for instance, preliminary experiments can be planned and conducted with miniaturized counterparts. We maintain such a fleet of scaled autonomous and cooperative vehicles using the Gulliver Testbed. Different use cases with our test-bed have successfully shown that it is possible to bridge between purely virtual experiments as carried out in simulations and physical experiments on real-scale proving grounds.

Our system design has two distinct parts that each has different timing properties, following the architectural hybridization concept. Given the uncertainties affecting the system operation and the confidence in the data used in control processes, we use the architectural concept of safety kernel. This concept is responsible for managing the task, in a way, that ultimately ensures the required safety goals. The vehicle limited ability to communicate prevents centralized solutions and open the door to cooperative ones. We consider sensory data that has validity attributes attached that defines that accuracy and conference in the data. The (decartelized) safety kernel uses these attributes to decide on a system service level that in turn will set the system performance level after cooperatively evaluating the service level. This version of the paper refers to the work that was done in KARYON with respect to local dynamic maps. We note that cooperation to construction of localization maps was earlier discussed in other projects, such as HIDENETS.

We have designed and implemented the Gulliver test-bed with an emphasis on demonstrating safety aspects of cooperative systems, and system architecture to the concrete implementation of fundamental components. The software architecture within each vehicle follows the proposed architectural pattern and, in particular, uses a safety kernel for safety management. For that, the hardware and software solution presented in this paper are based on an earlier design in which we have implemented and integrated the safety kernel in Gulliver vehicles [8]**.** Thus, the test-bed is adequate to demonstrate the architectural concept, and to show that it is possible to manage the performance level depending on the operational conditions while ensuring that the functions always perform safely.

In Annex F: reprint of the paper "**Driving with Confidence: Local Dynamic Maps That Provide LoS for the Gulliver Test-Bed**". C. Berger, O. Morales, T. Petig and E. M. Schiller. *Computer Safety, Reliability, and Security*, Springer, 2014, pp. 36-45.

5.2 Distributed Management and Representation of Environment Data and Context in Robotic Applications

The higher level of the KARYON middleware provides components for representing the environment. This part of middleware is designed for interpreting sensor information and for distributing information about the environment as provided by distributed stationary and mobile sensor systems available from road side infrastructure and floating car data. The respective component has been described in the deliverables D2.4 and D2.6. The general approach presented in the paper in Annex G is an extension that addresses storage and retrieval of environment information. The scheme allows to query and access data in smart environments, based on a two-step approach.

In a first step data is classified and organized according to a global and hierarchical structure. Based on CassandraDB and its ROS integration, described in [9], we developed a virtual overlay database for smart environments that links measurement data, robot and sensor descriptions, with location information. A first prototype was made available at:

http://svn.code.sf.net/p/ivs-ros-pkg/code/trunk/glodel/

This virtual overlay database is used to recreate local environment models and views. The generated complex geometric local environment models can be updated with available real-time data and furthermore be applied to extract more application specific information. For this purpose we developed a new kind of programming/querying language "SelectScipt", which is based on a SQL-like semantics. A first prototype was made available at:

https://pythonhosted.org/SelectScript_OpenRAVE/

SelectScript is a scripting language and can be executed at runtime and allows requesting facts about the environment and extracting further specific representations, e.g. such as views, sub models, or maps.

In Annex G: reprint of the paper "**Distributed management and representation of data and context in robotic applications**". A. Dietrich, S. Zug, S. Mohammad and J. Kaiser. *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 2014.

References

[1] Y.-H. Wei, Q. Leng, S. Han, A. K. Mok, W. Zhang and M. Tomizuka, "RT-WiFi: Real-time high-speed communication protocol for wireless cyber-physical control applications," in *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, 2013.

[2] Y. Xue, B. Ramamurthy and M. C. Vuran, "SDRCS: A service-differentiated real-time communication scheme for event sensing in wireless sensor networks," *Computer Networks,* vol. 55, no. 15, pp. 3287-3302, 2011.

[3] X. Zhu, S. Han, P.-C. Huang, A. K. Mok and D. Chen, "Mbstar: A real-time communication protocol for wireless body area networks," in *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, 2011.

[4] J. L. Souza and J. Rufino, "Analysing and reducing network inaccessibility in IEEE 802.15. 4 wireless communications," in *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, 2013.

[5] I. 802.15.4, *Part 15.4: Wireless Medium Access Control (MAC) And Physical Layer (PHY) Specifications For Low-Rate Wireless Personal Area Part 15.4: Wireless Medium Access Control (MAC) And Physical Layer (PHY) Specifications For Low-Rate Wireless Personal Area Networks (WPANs)Networks (WPANs),* 2011.

[6] *Wireless Access in Vehicular Environments - IEEE Standard 802.11p.*

[7] G. Combs, "The Wireshark network protocol analyser," Accessed in June 5, 2013.

[8] C. Berger, O. Morales, T. Petig and E. M. Schiller, "Driving with Confidence: Local Dynamic Maps That Provide LoS for the Gulliver Test-Bed," in *Computer Safety, Reliability, and Security*, Springer, 2014, pp. 36-45.

[9] A. Dietrich, S. Mohammad, S. Zug and J. Kaiser, "ROS Meets Cassandra: Data Management in Smart Environments with NoSQL," in *Proc. of the 11th International Baltic Conference (Baltic DB&IS 2014)*, Tallinn, Estonia, 2014.

[10] T. Brade, G. Jäger, S. Zug and J. Kaiser, "Sensor-and Environment Dependent Performance Adaptation for Maintaining Safety Requirements," in *Computer Safety, Reliability, and Security*, Springer, 2014, pp. 46-54.

[11] J. L. Souza and J. Rufino, "The Wi-STARK Architecture For Resilient Real-Time Wireless Communications".

[12] A. Dietrich, S. Zug, S. Mohammad and J. Kaiser, "Distributed management and representation of data and context in robotic applications," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 2014.

[13] B. V. R. J. a. H. E. Daniel Skarin, "A Fault-Injection Prototype for Safety Assessment of V2X Communication," in *DEPEND 2014 : The Seventh International Conference on Dependability*, 2014.

Annex A The Wi-STARK Architecture For Resilient Real-Time Wireless Communications

J. L. Souza and J. Rufino, “The Wi-STARK Architecture For Resilient Real-Time Wireless Communications”.

This page is intentionally left blank.

The Wi-STARK Architecture For Resilient Real-Time Wireless Communications[*]

Jeferson L. R. Souza
jsouza@lasige.di.fc.ul.pt

José Rufino
jmrufino@ciencias.ulisboa.pt

Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, Portugal
Laboratório de Sistemas Informáticos de Grande-Escala (LaSIGE)
Navigators Research Team

ABSTRACT

Networking communications play an important role to secure a dependable and timely operation of distributed and real-time embedded system applications; however, an effective real-time support is not yet properly addressed in the wireless realm. This paper presents *Wi-STARK*, a novel architecture for resilient and real-time wireless communications within an one-hop communication domain. Low level reliable (frame) communications, node failure detection, membership management, and networking partition control are provided; since these low level services extend and build upon the exposed interface offered by networking technologies, *Wi-STARK* is in strict compliance with wireless communication standards, such as IEEE 802.15.4 and IEEE 802.11p. The *Wi-STARK* service interface is then offered as operating system primitives, helpful for building distributed control applications. The one-hop dependability and timeliness guarantees offered by *Wi-STARK* are a fundamental step towards an effective design of real-time wireless networks with multiple hops, including end-to-end schedulability analysis of networking operations.

Categories and Subject Descriptors

C.4 [**Computer System Organisation**]: [Fault tolerance]; C.3 [**Special-Purpose and Application Based Systems**]: Real-time and embedded systems; C.2.1 [**Computer Communication Networks**]: Network Architecture and Design—*Wireless communication*

Keywords

wireless communications, real-time, dependability, timeliness, resilience, fault tolerance, wireless sensor and actuator networks, Wi-STARK

1. INTRODUCTION AND MOTIVATION

Advances in microelectronics enable the development and integration of networking computing systems in environments with different levels of criticality, monitoring and controlling physical entities such as nuclear reactors, physical structure of buildings and bridges, and power grids. In these kind of environments, usually known as Cyber Physical Systems (CPS), communications may have safety-critical constrains, implying a mandatory provision of real-time communication guarantees to secure the dependable and timely operation of the entire system.

The literature addressing real-time support on the wireless realm can be classified into two distinct domains: (*a*) communication protocols and architectures, and (*b*) schedulability analysis.

The contributions to real-time communication protocols and architectures, such as [16, 17, 18], are concerned with the provision of end-to-end guarantees within multiple hop networks. However, some of them require strong assumptions with respect a global notion of time (synchronised clocks among all nodes of a multiple hop network), which is a problem by itself without an easy solution. Furthermore, the used error model only assumes the loss of data frames, neglecting the effects that control frame errors may have on the operation of the Medium Access Control (MAC) sublayer, which may generate network partitions during long periods of time. These partitions may imply an unpredictable temporal behaviour and thus those protocols and architectures may, at the best, only provide probabilistic real-time guarantees.

The schedulability analysis of wireless networking communications [3, 11, 12] aims to verify if all transmissions can meet their deadlines for a given traffic workload, considering the end-to-end temporal guarantees wanted for a target network. Such end-to-end guarantees depend on the real-time guarantees secured within each single hop. Single hop guarantees can, on its turn, be derived from the temporal behaviour provided by the networking technology (communication protocols included), which must take into account the expected error conditions.

Conjugating dependability and real-time message delivery guarantees with wireless communications is a difficult problem. Instead of following the classic approach described in the wireless communication literature, and trying to estab-

[*] This work was partially supported by the EC, through project IST-FP7-STREP-288195 (KARYON); by FCT/DAAD, through the transnational cooperation project PROPHECY; and by FCT, through project PTDC/EEI-SCR/3200/2012 (READAPT) and through LaSIGE Strategic Project PEst-OE/EEI/UI0408/2014.

lish those guarantees end-to-end —using a traditional point-to-point communication model —we take a divide to conquer approach, which is motivated by the following statement:

> *If no real-time guarantees can be offered within communications at one-hop of distance, no real-time guarantees can be offered within multiple hop communications at all.*

That means, any dependable real-time message delivery guarantee has to be secured first within the one-hop of distance wireless space, prior to be extended end-to-end, across multiple hops. Thus, this paper presents a design overview of a novel wireless communications architecture dubbed *Wi-STARK*, which has three main goals: (1) taking advantage of the intrinsic broadcast properties of the shared wireless communication medium within one-hop space, (2) providing dependability and real-time guarantees within such one-hop space, and (3) ensuring the feasibility of end-to-end schedulability analysis given the bounded transmission delay guarantees within each single hop. The *Wi-STARK* design is compliant with wireless communications standards, being able to offer at the lowest level of communications a set of useful and semantically rich services such as reliable and timely communications, node failure detection, membership management, and networking partition control. Since these services are built upon the exposed interface offered by current networking technologies, the *Wi-STARK* architecture can be easily implemented using Commercial Off-The-Shelf (COTS) components. The *Wi-STARK* service interface can easily be made available at the operating system Application Programming Interface (API).

To present the details concerning the design of the *Wi-STARK* architecture, this paper is organised as follows: section 2 presents a brief description of the system model, which is the foundation for the design of the *Wi-STARK* architecture; section 3 presents the main components and characteristics of the *Wi-STARK* architecture; section 4 presents the primitives and semantics of the *Wi-STARK* service interface; and finally, section 5 presents the conclusion and future directions of the design and applicability of the *Wi-STARK* architecture.

2. SYSTEM MODEL

All networking communications described in this paper are performed within the scope of a physical and data link layer abstract networking model dubbed Wireless network Segment (WnS), which establishes a broadcast domain where all wireless nodes are one-hop of distance from one another. *This simple approach empowers the achievement of a first and fundamental result: the capability of exploiting the broadcast nature of the shared one-hop communication space.*

The formalisation of the WnS is expressed by a 4-Tuple, $WnS \overset{def}{=} \langle X, x_m, C, W \rangle$, where X is the set of wireless nodes members of the WnS; x_m is the WnS coordinator, $x_m \in X$; C represents a set of radio frequency (RF) communication channels; and W represents the set of networking access protocols utilised in the support of frame transmissions. As illustrated in the graphical representation of Fig. 1, the intersection of the communication range of all nodes within the WnS constitutes its broadcast domain, where each node $x_j \in X$ is able to sense any transmission from any other node $x_q \in X$.

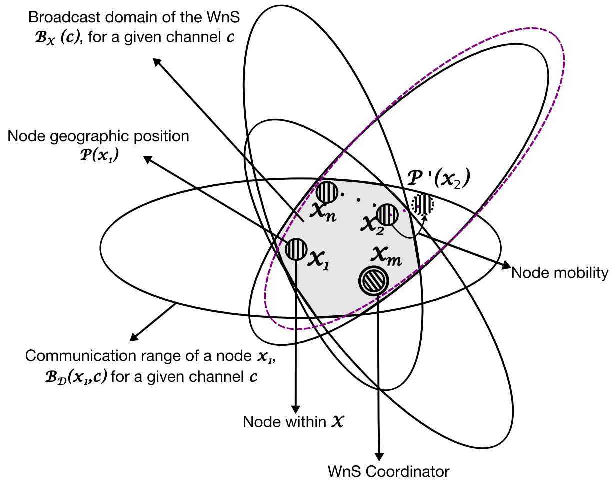

Broadcast domain of the WnS
$\mathcal{B}_X$ (c), for a given channel c
Node geographic position
$\mathcal{P}(x_1)$
$\mathcal{P}'(x_2)$
x_n
x_2
x_1
x_m
Node mobility
Communication range of a node x_1,
$\mathcal{B}_D(x_1,c)$ for a given channel c
Node within X
WnS Coordinator

Figure 1: The Wireless Network Segment (WnS) abstraction

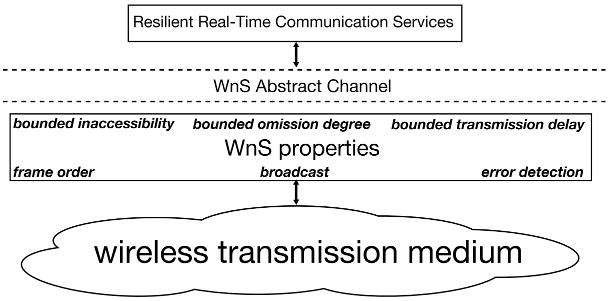
2.1 Fault Model

The failure of a networking component (a channel $c \in C$ or a node $x \in X$) is identified using an omission fault model, where frame errors are transformed into omissions. The occurrence of frame errors may be originated by disturbances caused by the presence of electromagnetic interferences on the communication channel, or malfunction within the node machinery, being accounted as omissions for the purpose of monitoring networking components.

For each received frame, each node $x \in X$ locally accounts observed omissions. When the number of observed omissions exceeds the component's omission degree bound, f_o, the failure of such component can be locally signed. Errors occurred at the wireless communication medium may affect only some nodes, which implies omissions may be accounted inconsistently at the different nodes of the WnS.

Both omissions with origin in the channel and at the channel end-points (i.e., the nodes) are accounted for. When successive frames are received with errors from a given channel input — i.e. a node $x \in X$ — exceeding a given omission degree bound, a *node persistent failure* is detected and signalled; when no traffic is received from node $x \in X$ within a bounded monitoring time interval, a *node crash failure* is detected and signalled.

Each node $x \in X$ may also inconsistently experience a temporary loss of connectivity with the WnS, caused by a phenomenon dubbed network inaccessibility [13]. A period of network inaccessibility may be induced by glitches in the MAC sublayer operation, such as those that may result from the omission of a MAC control frame (e.g., beacon). The network cannot be considered failed; it only enters into a temporary state where the communication service is not provided to some or all of the nodes. The loss of connectiv-



WnS1 - ***Broadcast***: correct nodes, receiving an uncorrupted frame transmission, receive the same frame;

WnS2 - ***Frame Order***: any two frames received at any two correct nodes are received in the same order at both nodes;

WnS3 - ***Error Detection***: correct nodes *detect and signal* any corruption done during frame transmissions in a locally received frame;

WnS4 - ***Bounded Omission Degree***: in a known time interval $\mathcal{T}_{rd}$, omission failures may occur in at most k transmissions;

WnS5 - ***Bounded Inaccessibility***: in a known time interval $\mathcal{T}_{rd}$, a wireless network segment may be inaccessible at most i times, with a total duration of at most $\mathcal{T}_{ina}$;

WnS6 - ***Bounded Transmission Delay***: any frame transmission request is transmitted on the WnS, within a bounded delay $\mathcal{T}_{td} + \mathcal{T}_{ina}$.

Figure 2: WnS abstract channel properties

ity due to transient node mobility is also treated under the inaccessibility model.

Mobility may drive nodes to outside of the WnS, as illustrated in Fig. 1, where node x_2 using channel c moves from the geographic position $P(x_2)$ to the geographic position $P'(x_2)$. In despite of x_2 transmissions at the new position may reach all nodes of the WnS, the transmissions from the WnS coordinator, $x_m \in X$, do not reach node x_2 at position $P'(x_2)$. The permanent mobility of a node to outside of the WnS broadcast domain is then transformed into a *node crash failure* in our fault model.

2.2 WnS abstract channel properties

Communications at the lowest levels of the networking protocol stack can be abstracted by a set of correctness, dependability, and timeliness properties, which are not dependent on any particular networking technology. In the context of the WnS model such properties are seen as being provided by a single abstract communication channel dubbed WnS abstract channel, as illustrated in Fig. 2.

Property WnS1 (*Broadcast*) formalises that it is physically impossible for a node $x \in X$ to send conflicting information (in the same broadcast) to different nodes, within the broadcast domain of the WnS [2], $B_X(c)$, for a given channel $c \in C$ (see Fig. 1).

Property WnS2 (*Frame Order*) is common in network technologies (wireless technologies included), being imposed by the wireless communication medium of each channel $c \in C$, and resulting directly from the serialisation of frame transmissions on the shared wireless communication medium.

Property WnS3 (*Error Detection*) has both detection and signalling facets; the detection facet, traditionally provided by classical MAC sublayers, derives directly from frame protection through a frame check sequence (FCS) mechanism, which most utilised algorithm is the cyclic redundancy check (CRC); the signalling facet is provided by the FCS extension introduced in [15], which is able to signal omissions detected in frames received with errors. No fundamental modifications are needed to the wireless MAC standards, such as IEEE 802.15.4 [8]. The use of such unconventional extension is enabled by emerging controller technology, such as reprogrammable technology and/or open core MAC sublayer solutions, which are present, for example, in the development kits from ATMEL [1]. With the CRC polynomials used in wireless MAC sublayers, the residual probability of undetected frame errors is negligible [4, 5].

Property WnS4 (*Bounded Omission Degree*) formalises for a channel, $c \in C$, the failure semantics introduced earlier in the fault model definition, being the abstract channel omission degree bound, $k \geq f_o$. The omission degree of a WnS abstract channel can be bounded, given the error characteristics of its wireless transmission medium [4, 9, 13].

The *Bounded Omission Degree* property is one of the most complex properties to secure in wireless communications. Securing this property with optimal values and with a high degree of dependability coverage may require the use of multiple RF channels. In [15] we have advanced on how this can be achieved by monitoring channel omission errors, and switch between RF channels upon detecting the channel omission degree bound has been exceeded.

The time domain behaviour of a WnS is described by the remaining properties. Property WnS6 (*Bounded Transmission Delay*) specifies a maximum frame transmission delay, which is $\mathcal{T}_{td}$ in the absence of faults. The value of $\mathcal{T}_{td}$ includes the medium access and transmission delays and it depends on message latency class and overall offered load bounds [6, 10]. The value of $\mathcal{T}_{td}$ does not include the effects of omission errors. In particular, $\mathcal{T}_{td}$ does not account for possible frame retransmissions. However, $\mathcal{T}_{td}$ may include extra delays resulting from longer WnS access delays derived from subtle side-effects caused by the occurrence of periods of network inaccessibility [13]. Therefore, the bounded transmission delay includes $\mathcal{T}_{ina}$, a corrective term that accounts for the worst case duration of inaccessibility glitches, given the bounds specified by property WnS5 (*Bounded Inaccessibility*). The inaccessibility bounds depend on, and can be predicted by the analysis of MAC sublayer characteristics [13].

3. THE Wi-STARK ARCHITECTURE

The *Wi-STARK* is a new low level architecture that takes advantage of the intrinsic broadcast property of the shared wireless communication medium, and of the set of correctness, ordering, dependability, and timeliness properties offered by the WnS abstraction (Section 2.2) to establish a robust, resilient and real-time one-hop communication domain for wireless networks.

The *Wi-STARK* architecture design is open and flexible, being composed by two layers dubbed *Channel Layer* and

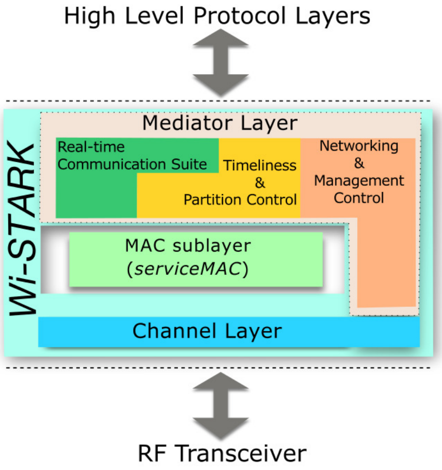

High Level Protocol Layers
Wi-STARK
Mediator Layer
Real-time Communication Suite
Timeliness & Partition Control
Networking & Management Control
MAC sublayer (serviceMAC)
Channel Layer
RF Transceiver

Figure 3: The *Wi-STARK* Architecture

Mediator Layer. As shown in Fig. 3, these layers are by design wrapping the standard MAC sublayer to improve: the control and use of RF communication channels; and, the services offered to high level protocol layers.

3.1 Channel Layer

The *Channel Layer* (Fig. 4) is a thin layer that provides a common interface to transparently control the use of a given RF communication channel $c \in C$ for purposes of frame transmission and reception, incorporating useful extensions to enhance the dependability of communications. A RF communication channel $c \in C$ is an abstract representation of the wireless transmission medium plus a piece of hardware dubbed RF transceiver, which conjugates a residual part of the MAC sublayer, herein called, *basicMAC* and the physical (PHY) layer itself.

The *Channel Layer* extends the *basicMAC* to exploit the exposed RF transceiver interface, and the parametrisation features thereof. In particular, the *Channel Layer* implements: the FCS extension (specified in [15]), which secures the WnS3 property of the WnS; the accounting of channel omissions and the detection of a RF communication channel failure, upon exceeding the omission degree bound, k (accordingly with WnS4); the RF communication channel switch strategy specified in [15].

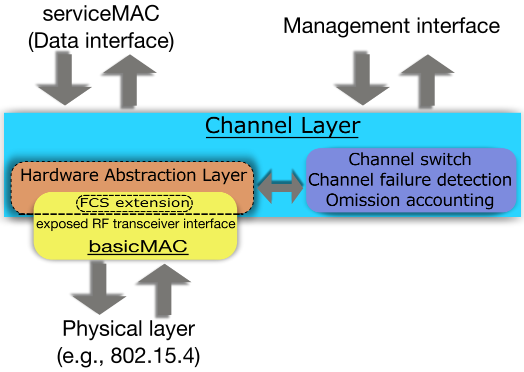

serviceMAC (Data interface)
Management interface
Channel Layer
Hardware Abstraction Layer
Channel switch
Channel failure detection
Omission accounting
FCS extension
exposed RF transceiver interface
basicMAC
Physical layer (e.g., 802.15.4)

Figure 4: Channel Layer

3.2 MAC Sublayer: serviceMAC

The MAC sublayer illustrated in Fig. 3 is the standard MAC sublayer present in the traditional wireless networking protocol stack, such as those specified within the IEEE 802.15.4 [8] and IEEE 802.11p [7] wireless standards. In the context of the *Wi-STARK* architecture such standard MAC sublayer is dubbed *serviceMAC*, offering only conventional unreliable data frame and management service interfaces. No modifications are needed for its integration in the *Wi-STARK* architecture. In this sense, the *Wi-STARK* architecture is highly flexible supporting the integration of any MAC sublayer, including the real-time variants proposed in [16, 17].

3.3 Mediator Layer

The *Mediator Layer* is an extensible sublayer, specially designed to mediate the communication flow from (and to) the high level protocol layers, as illustrated in Fig. 3. The *Mediator Layer* is responsible for the semantically rich service interface offered by *Wi-STARK*, effectively augmenting the services offered by the standard MAC sublayer. Three main components compose the *Mediator Layer*: the Real-Time Communication Suite, the Timeliness & Partition Control, and the Networking & Management Control.

3.3.1 Real-time Communication Suite

The Real-Time Communication Suite (RTCS) is the component responsible for the data communication services offered by the *Wi-STARK* architecture, as illustrated in Fig. 5. The RTCS includes a *Message Request Dispatcher* that forwards any high level message transmit request to the adequate instance of the RTCS protocol bundle. Messages submitted at the *Wi-STARK* service interface have a maximum length for allowing the encapsulation of their content in exactly one frame, without necessity of fragmentation.

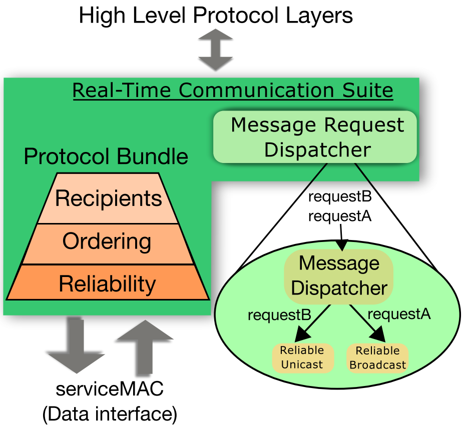
The table of Fig. 5 specifies the fundamental properties (recipients, ordering, and reliability) characterising the different variants of the protocols to be included in the RTCS protocol bundle. For example: a totally ordered reliable message delivery targeting all correct nodes features the well known *atomic broadcast* primitive. This specification is open and extensible: other attributes (e.g., *temporal order*) and other properties (e.g., *urgency*) can be included.

The *Wi-STARK* architecture design provides two fundamental guarantees to the high level protocol layers and applications:

Temporal-bounded communications: every transmitted message[1] is successfully received by all relevant correct nodes of the WnS within a known temporal bound, $\mathcal{T}_{Tx-Data}$.

The value of $\mathcal{T}_{Tx-Data}$ is directly derived from the combination of four important properties of the WnS: WnS3 (*Error Detection*), WnS4 (*Bounded Omission Degree*), WnS5 (*Bounded Inaccessibility*), and WnS6 (*Bounded Transmission Delay*). In the absence of errors, the *Wi-STARK* protocols execute in a single round and the upper bound for all correct nodes of the WnS receiving a message successfully is: $\mathcal{T}^{wc-ne}_{Tx-Data} = 2.\mathcal{T}_{td}$; being $\mathcal{T}_{td}$ the maximum frame transmission delay in the absence of errors.

[1] A message is a high level protocol layer data service unit.



Real-Time Communication Suite	
Property	Attributes

Recipients	Single node (Unicast);
	Multiple nodes (Multicast);
	All nodes (Broadcast)

Ordering	Unordered; Totally ordered
Reliability	Unreliable; Reliable

Figure 5: Real-Time Communication Suite

In the presence of errors, frames[2] may have to be retransmitted and the protocols within the *Wi-STARK* architecture may require more than one round to be executed, up to a limit given by $k+i+1$ (as specified by properties WnS4 and WnS5); all relevant correct nodes can successfully receive **any message transmitted with any reliable communication protocol** provided by the *Wi-STARK architecture* in, at most, $\mathcal{T}^{wc}_{Tx-Data} = (k+i+1) \times (2.\mathcal{T}_{td}) + \mathcal{T}_{ina}$. The timer utilised by reliable protocols to control protocol execution is configured with its optimal value (i.e., $\mathcal{T}_{td}$), and extended (if needed) by the real value of the network inaccessibility, t_{ina}, adding up to at most $\mathcal{T}_{ina}$ [14].

A failure of the RF communication channel in use is detected by the violation of k, the channel omission degree bound (WnS4), being the *Wi-STARK* architecture able to switch to another channel to keep the networking communications operational; the duration of the "communication blackout" resultant from that *channel failure* is then incorporated in the network inaccessibility model through $\mathcal{T}_{ina}$.

Message delivery: every transmitted message is delivered to all relevant correct nodes of the WnS.

Message delivery guarantees emerge from reliable communication protocols of the *Wi-STARK* architecture, which exploit the nature of the shared wireless communication medium (properties WnS1 and WnS2) to offer *totally ordered delivery* guarantees.

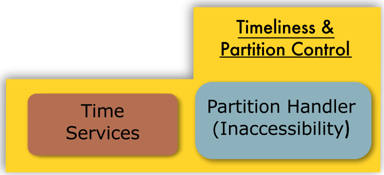

Timeliness & Partition Control
Time Services
Partition Handler (Inaccessibility)

Figure 6: Timeliness & Partition Control

3.3.2 Timeliness & Partition Control

The *Timeliness & Partition Control* (TPC) presents the transversal components that deals with the temporal aspects of the service offered by the *Wi-STARK* architecture. As shown in Fig. 6, the TPC component incorporates *Time Services* that include the management of protocol timers and other services used in the temporal control of *Wi-STARK* components.

The *Partition Handler* is focused to detect the occurrence, and to be aware of any partitioning incidents caused by the presence of periods of network inaccessibility. Controlling networking inaccessibility allows the use of optimal timeout values, which are automatically extended [14] when a period of inaccessibility occurs, preventing the propagation of premature timeout errors to other components and to high protocol layers.

3.3.3 Networking & Management Control

The *Networking & Management Control* component (illustrated in Fig. 7) incorporates all the functionalities of the *Mediator Layer* responsible for managing the dependable operation of each node $x \in X$. The management responsibilities assigned to the *Mediator Layer* include controlling all internal configuration of the *Wi-STARK* architecture, the parameters of the MAC sublayer (*basicMAC* and *serviceMAC* included), and the provision of management services to support the WnS formation.

All configurations can be performed statically or dynamically. The static configuration is target for hard real-time environments where all analyses of the traffic pattern, error conditions, and mobility models are performed offline, being stored in the *Wi-STARK Information Base* (Fig. 7). The *Mediator Layer* (self-)adaptation and dynamic configuration capabilities are related with mixed-critical and soft real-time requirements, which are outside the scope of this paper.

The membership and node failure detection offered by the *Mediator Layer* were designed to control and establish a consistent view of all members of the WnS, which is represented by the abstract set, X.

4. Wi-STARK DATA SERVICE INTERFACE

In the perspective of networking protocol developers, the dependability and timeliness guarantees offered by the *Wi-STARK* architecture are represented by a set of fundamental primitives for transmission and reception of messages to/from the network, which are specified in Table 1.

All of the primitives present in the *Wi-STARK* data service

[2] A frame is the MAC sublayer protocol data unit.

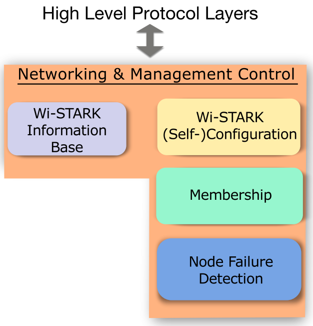

High Level Protocol Layers
Networking & Management Control
Wi-STARK Information Base
Wi-STARK (Self-)Configuration
Membership
Node Failure Detection

Figure 7: Networking & Management Control

Wi-STARK data service interface	
Primitives	**Description**
MLA.Data.request	Requests a message transmission using one of the *Wi-STARK* communication protocols.
MLA.Data.confirm	For reliable services, it confirms message delivery at recipients. Otherwise, it confirms only message transmission.
MLA.Data.indication	Notifies the arrival of a message.

Table 1: *Wi-STARK* data service interface

interface are easily integrated into embedded and real-time operating systems, being available as system calls associated to the wireless networking protocol stack.

5. CONCLUSION

This paper presented the architectural design of *Wi-STARK*, a novel low level architecture for resilient and real-time one-hop wireless communications. The definition of *Wi-STARK* is based on the establishment of an abstract communication model dubbed Wireless network Segment (WnS), which offer a set of correctness, dependability, and timeliness properties to support the design of resilient communication services for wireless networks.

Wi-STARK is compliant with wireless standards such as IEEE 802.15.4 and IEEE 802.11p, being capable to offer support for low level reliable message communication, node failure detection and membership, and networking partition control. Future directions involves the incorporation of the Wi-STARK service interface in the API of embedded real-time operating systems, and the extension of one-hop guarantees for multi-hop networking scenarios.

6. REFERENCES

[1] ATMEL Coorporation. *IEEE 802.15.4 MAC Software Package - User guide*, May 2012.

[2] O. Babaoğlu and R. Drummond. Streets of Byzantium: Network Architectures for Fast Reliable Broadcasts. *IEEE Trans. on Soft. Engineering*, SE-11(6), June 1985.

[3] O. Chipara, C. Lu, and G.-C. Roman. Real-Time Query Scheduling for Wireless Sensor Networks. *IEEE Trans. on Computers*, 62(9), September 2013.

[4] D. Eckhardt and P. Steenkiste. Measurement and Analysis of The Error Characteristics of An In-Building Wireless Network. In *2nd SIGCOMM Conference*, 1996.

[5] T. Fujiwara, T. Kasami, A. Kitai, and S. Lin. On The Undetected Error Probability for Shortened Hamming Codes. *IEEE Trans. on Comm.*, 33(6), June 1985.

[6] M. Hameed, H. Trsek, O. Graeser, and J. Jasperneite. Performance Investigation And Optimization of IEEE 802.15.4 For Industrial Wireless Sensor Networks. In *IEEE 13th ETFA Conference*, September 2008.

[7] IEEE 802.11p. Wireless Access in Vehicular Environments - IEEE Standard 802.11p, 2010. Amendment to IEEE Standard 802.11-2007.

[8] IEEE 802.15.4. Part 15.4: Wireless Medium Access Control (MAC) And Physical Layer (PHY) Specifications For Low-Rate Wireless Personal Area Networks (WPANs) - IEEE standard 802.15.4, 2011.

[9] M. Petrova, J. Riihijarvi, P. Mahonen, and S. Labella. Performance Study of IEEE 802.15.4 Using Measurements And Simulations. In *WCNC Conference*, Las Vegas, NV, USA, April 2006.

[10] I. Ramachandran, A. K. Das, and S. Roy. Analysis of The Contention Access Period of IEEE 802.15.4 MAC. *ACM Trans. on Sensor Networks*, 3, March 2007.

[11] A. Saifullah, Y. Xu, C. Lu, and Y. Chen. Priority Assignment For Real-Time Flows in WirelessHART Networks. In *23rd ECRTS Conference*, July 2011.

[12] W. Shen, T. Zhang, M. Gidlund, and F. Dobslaw. SAS-TDMA: A Source Aware Scheduling Algorithm For Real-Time Communication In Industrial Wireless Sensor Networks. *Springer Wireless Networks Journal*, 19(6), August 2013.

[13] J. L. R. Souza and J. Rufino. Characterization of inaccessibility in wireless networks - A Case Study on IEEE 802.15.4 Standard. In *IFIP 3th IESS Conference*, September 2009.

[14] J. L. R. Souza and J. Rufino. An Approach to Enhance The Timeliness of Wireless Communications. In *5th UBICOMM Conference*, Lisbon, Portugal, November 2011.

[15] J. L. R. Souza and J. Rufino. Analysing And Reducing Network Inaccessibility in IEEE 802.15.4 Wireless Communications. In *IEEE 38th LCN Conference*, Sydney, Australia, October 2013.

[16] Y.-H. Wei, Q. Leng, S. Han, A. Mok, W. Zhang, and M. Tomizuka. RT-WiFi: Real-Time High-Speed Communication Protocol For Wireless Cyber-Physical Control Applications. In *IEEE34th RTSS Conference*, December 2013.

[17] Y. Xue, B. Ramamurthy, and M. C. Vuran. SDRCS: A Service-Differentiated Real-Time Communication Scheme For Event Sensing in Wireless Sensor Networks. *Computer Networks*, 55(15), June 2011.

[18] X. Zhu, S. Han, P.-C. Huang, A. Mok, and D. Chen. MBStar: A Real-Time Communication Protocol For Wireless Body Area Networks. In *23rd ECRTS Conference*, July 2011.

Annex B Sensor- and environment dependent performance adaption for maintaining safety requirements

T. Brade, G. Jäger, S. Zug and J. Kaiser, "Sensor-and Environment Dependent Performance Adaptation for Maintaining Safety Requirements," in *Computer Safety, Reliability, and Security*, Springer, 2014, pp. 46-54. [10]

This page is intentionally left blank.

Sensor- and environment dependent performance adaptation for maintaining safety requirements

Tino Brade, Georg Jäger, Sebastian Zug, and Jörg Kaiser

Otto-von-Guericke-University of Magdeburg,
Institute for Distributed Systems (IVS),
Germany
(brade,jaeger,zug,kaiser)@ivs.cs.uni-magdeburg.de

Abstract. Driving assistance or automated driving depends to a large extent on the correct perception of the environment. Because automated driving functions have to be proven safe under all operational conditions, worst-case assumptions concerning the sensors and also the environment have to be assumed. In this paper, we propose a scheme that allows taking weaker assumptions. This is based on a continuous assessment of the quality of sensor data, a model of the interaction between the control process and the environment and the possibility to adapt the performance. We present an example of a car autonomously driving a simple course and adapting its speed according to the environment and the confidence in the perceived sensor data. We derive a set of simple safety rules used to adjust performance that, in the case given in the example affects the cruising speed.

1 Introduction

Functional safety is a non-debatable property for automotive systems. The requirements and procedures for functional safety are fixed in a respective standard [1]. At the same time, it is one of the most challenging tasks ensuring functional safety for sophisticated driver assistance systems or complex automatic driving functions. One of the reasons is that assurance means that a function has to be proven safe before it can be put into operation in a car. Consequently, it has to be proven safe at design time for all driving situations and failures that may occur during operation. This results in worst-case assumptions about the environment, the perception system and the control application. Particularly, the control application makes implicit assumptions about the quality of sensor input by tolerating acceptable error margins due to a robust design of the control algorithm. The correct functionality is usually carefully checked during the testing phase. Statically assuming worst-case conditions at design time has some undesirable consequences. Firstly, it leads to high costs of the sensors because lower cost sensors, although in most cases will comply with the requirements, cannot ensure this at any time during a mission. Secondly, because perception and control are tightly intertwined, the test only provides a validation for a specific set of sensors. If sensors have to be replaced, only the same or very similar sensors can be used. Finally, because a system has to show that safety is guaranteed under all conditions, the requirements will be overly strict.

In this paper we will present a scheme for adapting performance according to environmental conditions and erroneous sensor information without sacrificing safety. This scheme is based on a number of system functions that have been elaborated during the KARYON project [2].

1. The KARYON architecture defines a safety kernel that allows to put the system in different levels-of-performance according to a set of safety rules. The safety rules specify the conditions in terms of system health state and the quality of sensor data.
2. Separating the design of the perception system from the design of the control application. The main advance over other systems is that on the control application side, we present a way, how the control application can specify the quality of sensor data explicitly. On the sensor side, we provide a concept, how to quantify the confidence in sensor data. Instead of having to validate a single sensor-control block, we can validate these blocks separately. This allows dealing with a changing set of sensors easily. Additionally, it is a prerequisite when assuming remote sensors that are not known at design time. A more detailed discussion about this point can be found in [3].
3. The scheme to quantify the confidence in sensor data called "validity" allows the dynamic assessment of sensor data during run-time. Because the KAYON safety kernel allows to react if the validity is too low, we are able to handle this situation dynamically. The example below explains this in more detail.
4. Knowledge about the road is exploited to define the tolerable error margins in which safety can be ensured. In our example, we simulate a simple course and build a model of the process that relates speed, validity of sensor data and the position of the car to define safe conditions for controlling the car

The contribution of this paper is firstly showing how the notion of validity can be used to express error margins and secondly how the knowledge about the course of a road, the speed of a car, its position and orientation can be described by a mathematical model allowing to determine the adequate level of performance for a given safety level.

The paper is organized as follows: In the next section we briefly introduce the notion of validity and its relation to a failure model. This is needed to understand how error margins are expressed by a validity. Chapter 3 provides introduction and evaluation of the simulation example. Chapter 4 discusses the results and related work and the summary in chapter 5 concludes the paper.

2 Assessing the quality of sensor data

Sensors deliver a continuous range of values and often exhibit a subtle behaviour in case of external or internal disturbances. In our work, we use a data centric approach [4] to identify sensor failures, i.e. we try to infer faulty sensor data from

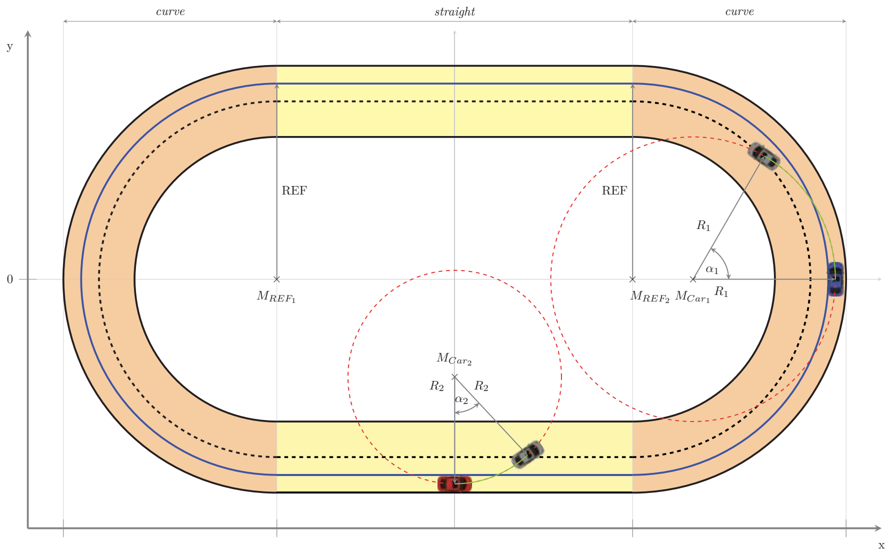


Fig. 1. Illustration of the environment

certain failure characteristics rather than applying space or time redundancy mechanisms. These parameters are derived empirically from testing a sensor exhaustively under many typical operational conditions and internal and external disturbances (e.g. voltage drops, electromagnetic glitches, intensive light for IR sensors and laser sensors etc.). A discussion of the resulting failure model can be found in [5]. Basically, we characterize a failure according to its amplitude and its occurrence probability. From these parameters, we calculate the anticipated validity of sensor data at design-time. A system performs at its best when no failures occur. With occurring sensor failures, additional robustness is required which has implications on the performance. The KARYON system allows to trade fading functionality against the level of system performance. The more failures will occur, the lower the performance that can be achieved safely by the system. For choosing an adequate performance level without violating safety, actual sensor data needs to be assessed at run-time. The key for achieving this is a consistent representation of the validity at design-time and at run-time. This allows proving the system to be safe for a given set of failures. By using the run-time validity, the system is switched to a performance level that is proven to cope with the occurred failures.

3 An automated driving scenario

This section considers a lane tracing application, which is a common task when building an autonomous car. Fig. 1 illustrates an example course for the automated car. In order to prove the system to be safe, we have to model the environment, specify system parameters and we make assumption regarding the operational context. This allows us to statically prove the compliance of our setup with requirements. Based on this prove, we identify conditions under which the car follows the line although sensor failures are present. Such conditions are expressed in terms of safety rules, which are used at run-time to choose the highest possible performance setting (LoS) of the application without risking safety. By analysing validity ranges at design-time we can derive safety rules, which are checked at run-time by the safety kernel. This is the basis to trigger switching the level of service (LoS).

The following example will explain how to derive these safety rules. For sake of clarity, we consider a simple course with straight and curved lanes. The course is modelled by the straight-line equation $REF = |y|$ and the circle equation $REF = \sqrt{(x-a)^2 + (y-b^2)}$. Clearly, such an environment model leads to an over-determined set of equations. For deriving safety rules, we have to assign system variables to define the basic kinematics of the car, to map the requirements and to make assumptions on which the system will be proven to be safe.

First, we assign system variables as follows: the steering angle (α) in a range of $-45°$ and $45°$, the highest maximum velocity of the car ($v \leq 15\frac{m}{s}$), the wheelbase of the car is set to $3m$ and the distance to the rear axle (a_2) assigned as $1.5m$. Theses system variables describe the kinematics of the car so that the anticipated position of the car can be calculated. Additionally, we have to define the sample time ($t = 0.5s$), which is the update rate of the observation and of the calculation of the steering command. This means that the perception-action loop of the car is periodically executed every 0,5 seconds.

Second, the kinematics of the car is required to calculate the impact of a certain steering angel on its position and orientation in the environment. For resolving this relation, we make use of the Ackerman condition: $R = \sqrt{a_2^2 + l^2 \cdot \cot^2(\alpha)}$. This condition allows us to calculate the deviation of the car from its ideal track assuming the current position and orientation of the car.

Third, in order to decide whether the deviation of the car from the ideal track is in line with requirements, we have to specify what is tolerable. In our scenario, we define a deviation of plus/minus one meter ($th_{err} \leq \pm 1m$) as acceptable.

Finally, we have to make assumptions in terms of the position and the orientation of the car with respect to the validity. In cases where the position sensor data has a high validity, the PID controller will keep the car very close to the ideal track. When sensor failures occur and the observed position is not accurate, the validity in sensor data drops and, as a result, the car deviates considerably in terms of position and orientation from the ideal track. The key for making such an analysis is the concept of design-time validity that specifies failure cases. This allows proving the system being safe under a set of assumptions. Without such assumptions, these check whether the system acts safe, could not be made at design time. This is because the equations representing the environment stays over-determined without assumptions regarding the quality of sensor inputs. Consequently, the combination of system variables, sensor inputs, and the controller output together with the environment model would be checked at

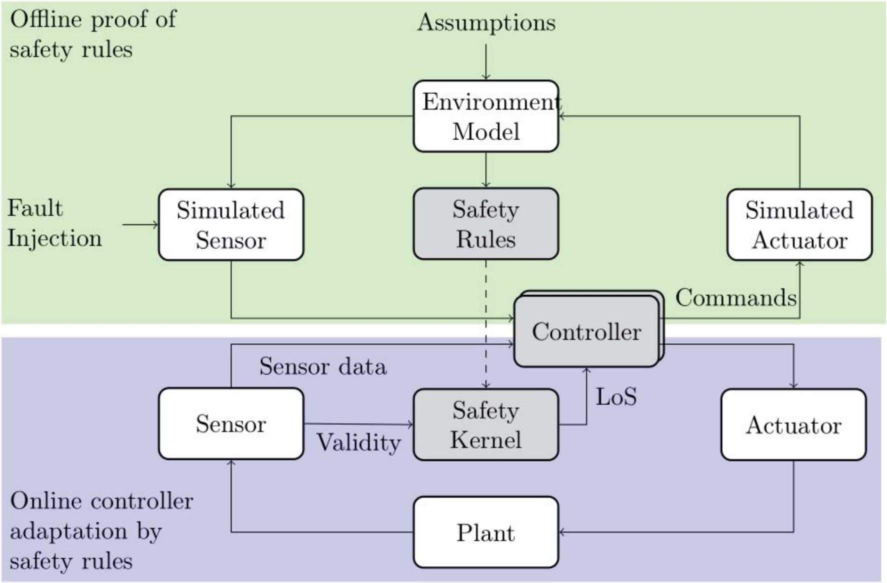


Fig. 2. Schematic overview of the concept

run-time. Obviously, run-time checking does not provide guarantees and so the response of the system is unknown when sensor failures occur.

3.1 Deriving safety rules at design-time

The objective of safety rules is to calculate the performance that can be reached without violating requirements. Fig. 3 depicts the problem of adjusting the steering angle based on an erroneous observation. C_1 shows the observed position of the car based on which a steering command is computed. By applying the steering command, the car drives to C_2. In case of sensor failures, the computed steering angle violates the requirement. The actual position now would be C_3 and the car moves to C_4 then. This shows the effect of erroneous observations. To maintain a safe behaviour we use the notion of sensor data validity to adapt the performance, which in this simple example means to adjust the velocity.

Safety rules define conditions under which the system operates safe. In our case, we observe the position of the car and compute a respective steering command in order to keep the car on track. As illustrated in Fig. 2, we derive safety rules by simulating the perception-action loop under a simple environment model. This setup allows us to check the compliance of the controller with requirements. It requires knowing the environment model, defining the system parameters and considering sensor failures that need to be handled at run-time. As shown in Fig. 2, the controller receives observations of a (simulated) sensor, which allows us to record the reaction of the controller on sensor failures that

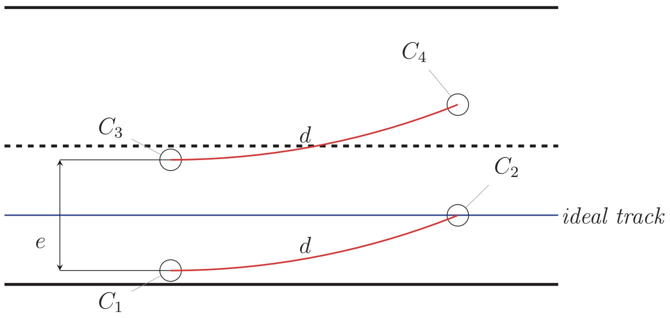


Fig. 3. The effect of sensor failures (e) without adapting the velocity (d)

have been injected. Consequently, we can check the controller response for every anticipated failure case. Based on this analysis we can define safety rules. Safety rules relate the validity of perception data to the level of service for the controller, which specifies system parameters, in our case the velocity of the car. The LoS may be translated to specific configurations or even different versions of the controller.

The following safety rules respect the validity of sensor data to calculate the highest velocity without violating requirements. Therefore, we compute the intersection point $I(x_i, y_i)$ between the requirement and the erroneous position, which is in fact unknown but estimated by the validity $C(x_c, y_c)$. By exploiting the time-space-relation $v = \frac{s}{t}$, we determine the distance of the actual position of the car to the intersection point I and so we receive the velocity for adapting the performance.

Safety rule for driving on the straight track In cases where the steering angle is zero ($\alpha = 0°$), the highest maximal velocity is then given by $v = \frac{|\overrightarrow{CI}|}{t}$. Otherwise, the car drives a circular path that is given by the Ackermann-condition ($R = \sqrt{a_2^2 + l^2 \cdot \cot^2(\alpha)}$) and correlates to the second safety rule.

Safety rule for driving on the curved track When driving a curve, we calculate the angle (γ) between the actual position of the car and the intersection point with the requirement: $\cos(\gamma) = \dfrac{\overrightarrow{MI} \times \overrightarrow{MC}}{|\overrightarrow{MI}| \cdot |\overrightarrow{MC}|}$ where M is the center of the steering cycle as shown in Fig. 1. The velocity that should not be exceeded, is therefore given by: $v = \dfrac{\pi \cdot R \cdot \frac{\gamma}{180°}}{t}$.

3.2 Checking safety rules at run-time

At run-time, detection mechanisms are used to assess the sensor data of the positioning sensor. The better an observation, the higher the validity of sensor

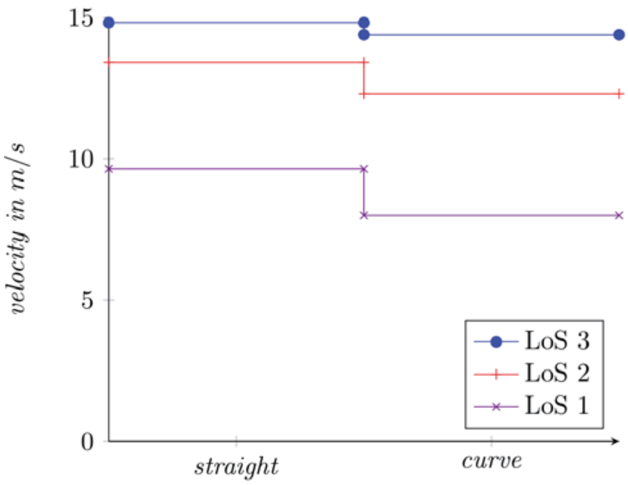


Fig. 4. Proven velocities of different performance levels to cope with positioning failures

data. As shown in Fig. 2, the safety kernel switches the controller into a level of service (LoS) dependent on the validity of actual sensor data. The safety rules thus specify the required validity for performing a certain level of service and serve as a decision basis to switch configurations of the controller. Whether the system operates safe by using this LoS setup was proven at design-time while deriving the safety-rules. This results in an approach that reduces the LoS in order to operate safe in case of sensor failures. Otherwise, the safety kernel switches to a higher LoS without jeopardizing safety.

4 Discussion

The proposed scheme is implemented in Simulink where V-Rep is used as for simulating the environment. We obtain reproducible results and are able to analyse the effect of observation failures on the performance of the system. By using a fault injection framework, we compare test cases with and without failures. In accordance to the injected failure amplitude, the car degraded its performance in order to comply with requirements. It should be noted that the system stops if the car is not able to keep within the lane due to injected failures or limitations of the steering angle.

In Fig 4, we plotted the resulting velocity of different LoS as a result of the derived safety rules. The blue curve (LoS 3) shows the highest performance level that can be reached when no failure occurs. Therefore, the velocities of the blue curve corresponds to the performance of an ideal system. The red curve labeled LoS 2 gives the velocity that can be set in cases where the positioning sensor suffers from noise. The violet curve (LoS 1) states the system performance when outliers and noise failures are considered. When making only worst case assumptions, the system performance would be statically set to the violet curve (LoS 1). In contrast our approach degrades the performance level only if necessary.

Comparing our approach to related work, we found approaches either dealing with sensor failures at design-time or at run-time only. Uncertainty margins [6] describe the characteristics of a sensor but they fail to distinguish failure types. The separation of failure types is provided by FMEA [7] that is limited to design-time analysis. On the other hand, confidence intervals [8], confidence classes [9] and also validates [10] provide a run-time representation but do no support design-time analysis. None of them can be used both at design-time as well as at run-time. Without such a consistent representation, the service levels of a system cannot statically proven safe at design-time and switched at run-time. On the side of an application, we found approaches [11], [12] for adapting the configuration in accordance to failures but such approaches ignore the operational context and the consideration of the environment. Nevertheless, those aspects are essential when proving the system to be safe.

5 Conclusion

The KARYON project developed an architectural pattern, which allows to react on a degraded functionality by switching to different levels of service, i.e. to differnt control schemes. The decision is based on the assessment of system health in a broad sense. In this paper, we focussed on failures of the sensor system. For a simple example we showed how the notion of validity can be used for design time analysis and also in run-time assessment of sensor data. A reliable safety kernel monitors the validity and takes actions if validity drops below a predefined bound. The bounds on validity and the necessary knowledge for the controller can be statically analysed at design time and transformed into safety rules to be executed at run-time for configuring controller functions.

Acknowledgment

This work has been supported by the EU under the FP7-ICT programme, through project 288195 Kernel-based ARchitecture for safetY-critical cONtrol (KARYON).

References

1. (ISO), *ISO 26262-1 to ISO 26262-9*, 1st ed., 2011.
2. A. Casimiro, J. Kaiser, E. M. Schiller, P. Costa, J. Parizi, R. Johansson, and R. Librino, "The karyon project: Predictable and safe coordination in cooperative vehicular systems," in *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on.* IEEE, 2013, pp. 1–12.

3. T. Brade, S. Zug, and J. Kaiser, "Validity-based failure algebra for distributed sensor systems," in *Reliable Distributed Systems (SRDS), 2013 IEEE 32nd International Symposium on*. IEEE, 2013, pp. 143–152.
4. K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, and M. Srivastava, "Sensor network data fault types," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 3, p. 25, 2009.
5. S. Zug, A. Dietrich, and J. Kaiser, "Fault-handling in networked sensor systems," *Fault Diagnosis in Robotic and Industrial Systems*, 2012.
6. R. J. Moffat, "Describing the uncertainties in experimental results," *Experimental thermal and fluid science*, vol. 1, no. 1, pp. 3–17, 1988.
7. D. H. Stamatis, *Failure Mode and Effect Analysis: Fmea from Theory to Execution*. Milwaukee,: ASQ Quality Press, 2003.
8. W. Elmenreich, "Fusion of continuous-valued sensor measurements using confidence-weighted averaging," *Journal of Vibration and Control*, vol. 13, no. 9-10, pp. 1303–1312, 2007.
9. H.-M. Piontek, *Self-description mechanisms for embedded components in cooperative systems*. Der Andere Verlag, 2007.
10. M. Duta and M. Henry, "The fusion of redundant seva measurements," *Control Systems Technology, IEEE Transactions on*, vol. 13, no. 2, pp. 173–184, 2005.
11. M. Blanke and J. Schröder, *Diagnosis and fault-tolerant control*. Springer, 2003, vol. 115.
12. P. M. Frank, "Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy: A survey and some new results," *Automatica*, vol. 26, no. 3, pp. 459–474, 1990.

Annex C A Tool for Real-Time Assessment Through Fault Injection in IEEE 802.15.4 Networks

R. P. Caldeira, J. L. R. Souza and J. Rufino, “A Tool for Real-Time Assessment Through Fault Injection in IEEE 802.15.4 Networks”, Submitted for publication

This page is intentionally left blank.

A Tool for Real-Time Assessment Through Fault Injection in IEEE 802.15.4 Networks

Rui Pedro Caldeira, Jeferson L. R. Souza, and José Rufino [*]

Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa
Lisboa, Portugal
{rcaldeira,jsouza}@lasige.di.fc.ul.pt,
jmrufino@ciencias.ulisboa.pt

Abstract. Advances in computer engineering and microelectronics have allowed the use of tiny and powerful computing platforms (i.e., sensors and actuators) everywhere, supporting the monitoring and control of, for example, process for industrial automation and functions within aerospace vehicles. Many of these systems have the ability to host, in the same computing platform, applications with different levels of criticality (or importance), i.e. mixed-critical systems. Wireless sensor and actuator networks (WSANs) become the vivid example of computer networks responsible for the monitoring and control activities of such systems, being the dependability, timeliness, and then the real-time properties of such networks crucial. One key point is that WSANs are extremely susceptible to communication errors induced by electromagnetic interferences. This paper presents a state of the art solution for the real-time assessment of WSANs in the presence of errors based on the IEEE 802.15.4 standard. The solution includes devices and functions to monitor the behaviour the network as well as methods to emulate accidental errors and to perform intentional attacks. All these resources are managed and controlled by a customised version of the well-known open-source Wireshark network protocol analyser. This allows the generation of network error reports fundamental to the evaluation of the real-time capabilities of current wireless network protocols and standards. These error reports contribute to a better knowledge of the error characteristics of WSANs and therefore enable the design of more robust and resilient solutions for WSANs operation.

1 Introduction and Motivation

Wireless Sensor and Actuator Networks are one of the latest revolutions in networking. The absence of cables stem a reduction of Size, Weight and Power Consumption (SWaP) which combined with node mobility, lead to the adoption of these networks as a fundamental communication platform of many different types of systems that may host applications with different levels of criticality (or importance), which are usually known as mixed-critical systems. Despite some advances in the support of real-time

[*] This work was partially supported by the EC, through project IST-FP7-STREP-288195 (KARYON); by FCT/DAAD, through the transnational cooperation project PROPHECY; and by FCT, through project PTDC/EEI-SCR/3200/2012 (READAPT) and through LaSIGE Strategic Project PEst-OE/EEI/UI0408/2014.

communication in wireless networks, there are still open problems that need the appropriate tools for their study and analysis. One key issue is that wireless networks are subjected to electromagnetic interferences from the surrounding environment that may impair ongoing communications. The presence of interferences can endanger the real-time guarantees of the communications as well as of the overall system. This paper discuses a highly flexible advanced tool that allows the real-time assessment of IEEE 802.15.4 networks through network monitoring and through emulation of accidental faults and injection of intentional attacks. The assessment of the network operation, and therefore the contributions of this paper, includes the facets that we detail next.

Network monitoring is a technique aimed at analysing the network interactions between its nodes and its characteristics such as reliability, timeliness and resilience. Network monitoring assumes a passive and non-intrusive role at network assessment. In particularly, it allows to assess normal network operation and the disturbances in its behaviour in the presence of errors.

Since particular error patterns may be specially relevant for the analysis of the network operation and their natural occurrence may be rare, there is the need to re-enact such error patterns, trough fault injection.

Our fault injection methodology allows to directly inject interferences in the wireless transmission medium with the objective to stress test the network. This may be achieved through the transmission of particular symbol patterns without obeying to the medium access control protocol in use. This translates to the injection of unrecognised noise. On the other hand, it may be useful to also inject properly formatted traffic. Our fault injection tool supports both. Furthermore, combining network monitoring with fault injection allows to trigger fault injection upon the occurrence of particular network traffic patterns. This coordinated action allows an active role in network assessment.

Network monitoring activities can be a very data-intensive task, specially when conducted concurrently with fault injection activities. Analysing the harvested data is simply too difficult to do without assistance. Wireshark [4], being a reference network protocol analyser, has the ability to create a graphical representation of network interactions, thus assisting in the network analysis task by being an integrated Graphical User Interface for the whole suite.

The remainder of the paper is structured as follows. Section 2 describes the related work. Section 3 describes the real-time assessment suite and its components. Section 4 describes the implementation details for the suite. Section 5 will walk through some simple use cases of this tool and finally section 6 describes the conclusion and future work.

2 Related Work

Monitoring and fault-injection are not new techniques. Both techniques are often used in conjunction to perform dependability evaluation of computer systems and networks.

All network monitors provide a very basic function. The hardware and additional features may vary but all capture network traffic. However, is usually restricted to correctly formatted packets. Existing tools traditionally do not capture and signal the occurrence of errors.

Due to WSANs growing popularity, researchers felt the need to create tools that could analyse such networks. Popular use cases for this type of devices are, for instance, those described in [8, 10]. However, these works focus in network performance evaluation, paying little to no attention to communication errors or to the stress of the network operation via fault injection campaigns.

With network security in mind, strategies for the injection of attacks in wireless networks were studied and prototypes were developed. Xu et al. [15] raised concerns about how insecure WSANs are. Wide range of attack strategies were defined in [15] taking into account the scarcity of power resources in wireless devices. On the other hand, O'Flynn [9], developed a dual-transceiver device capable of very precise attack injection techniques.

3 A suite for Real-Time Assessment of IEEE 802.15.4 Networks

In this section we will address in detail each component and how they all integrate following closely the architecture in Figure 1.

This tool for the real-time assessment of IEEE 802.15.4 networks is composed by four components. Firstly, with the purpose of capturing and report network traffic, a standalone network monitoring unit which assumes the role of packet sniffer was developed.

Secondly, to provide the ability to emulate accidental faults and to perform intentional attacks, a fault-injection unit has been built to flexibly support the injection of faults directly in the wireless transmission medium.

Thirdly, to provide interconnectivity support between the network monitoring unit and the fault injection unit, as well to support advanced network analysis and filtering functions, a special purpose hardware integration interface was introduced.

Finally, Wireshark, being a reference network protocol analyser with IEEE 802.15.4 packet visualization capabilities, was extended to manage all aspects of the network assessment, including control and data collection from the previously mentioned components.

3.1 Network Monitoring Unit

The Network Monitoring Unit includes a Commercial-Of-The-Shelf (COTS) hardware network interface device acting as networking monitoring probe (a.k.a. Sniffer) with the purpose of capturing all traffic transmitted through a specific channel. This is especially useful for testing networks and protocol implementations. The network monitoring probe is not an active member of the network, and is instead a passive listener. This is compliant with the IEEE 802.15.4 standard when using the so called Promiscuous Mode. In Promiscuous Mode, the network monitoring probe is allowed to capture and report any traffic it senses. Thus, the Promiscuous Mode is the key to capture all the frames sensed by the network monitoring probe. However, this is insufficient to capture the network error pattern since frames sensed with errors are traditionally silently discarded at very low levels of communications, by the network interface device hardware.

Wireless networks, due to the open nature of the transmission medium are, in general, extremely susceptible to electromagnetic interference (EMI) and therefore to the occurrence of frame errors. In order to capture also frames corrupted by errors, the operation of the network operating probe needs to be enhanced.

The IEEE 802.15.4 traffic is constituted by frames following the general format represented in Figure 2. The Frame Check Sequence (FCS) field is a 16-bit frame integrity number used to evaluate the correctness of each frame captured by the network monitoring probe. Any deviation from the original frame content, e.g. resulting from a corruption in the wireless transmission medium, is detected by the network monitoring probe. The residual probability of undetected frame errors is negligible [5].

A simple extension of this FCS integrity check mechanism [12], enabled by modern network interface controllers such as the Atmel AT86RF232 [1], allows to receive and signal erroneous frames. This is fundamental to evaluate the error characteristics of each network and can be used to estimate the quality of the communications of a specific network node or of the overall channel.

To enable a precise evaluation of network operation timing characteristics, a timestamp is generated at the network monitoring unit at the arrival of each frame, including erroneous frames. This timestamp is attached to the frame and delivered at the Hardware Integration Interface and to the Wireshark translator, as illustrated at Figure 1.

Concluding, the network monitoring unit offers to the surrounding components an extended promiscuous network traffic capture service that includes both correct and erroneous frames.

3.2 Fault-Injection Unit

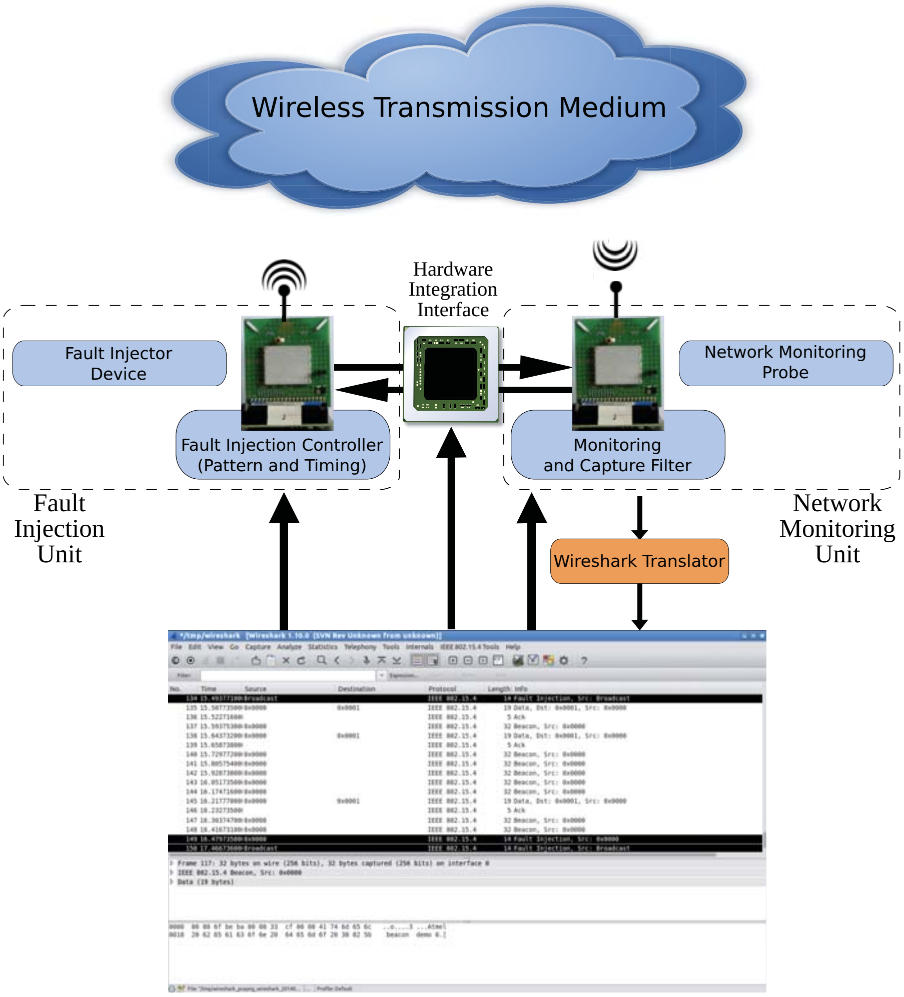
The Fault Injection Unit is constituted by a hardware network interface, the fault injection device, controlled by a software component, the fault injection controller, as illustrated in the left uppermost part of the Figure 1.

The fault injector device is a perfectly common Commercial-Of-The-Shelf (COTS) network interface with the exception that the network interface is configured to bypass the medium access control protocol thus allowing a direct access to the wireless transmission medium. Traditional IEEE 802.15.4 nodes use the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) mechanism to sense the medium before transmitting which is therefore disabled.

Disabling the medium access protocol allows effective fault injection actions. This way, the fault injector device supports the injection of user-defined: pure noise patterns; selected data patterns always preceded by the standard preamble[1];selected data patterns encapsulated in a correctly formatted frame, thus injecting a standard compliant transmission.

The Fault Injection Controller is a software component executing on top of the hardware fault injection device with the responsibility to control the patterns and timings to be utilized in a fault injection campaign.

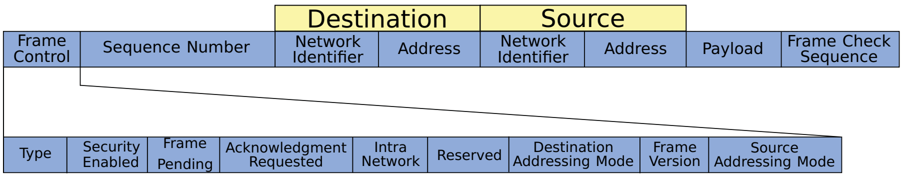
[1] The preamble is pre-defined sequence of symbols for synchronization of the receiver's circuitry with the incoming sequence of symbols.


Wireless Transmission Medium
Hardware Integration Interface
Fault Injector Device
Fault Injection Controller (Pattern and Timing)
Fault Injection Unit
Network Monitoring Probe
Monitoring and Capture Filter
Network Monitoring Unit
Wireshark Translator

```
*/tmp/wireshark  [Wireshark 1.10.0 (SVN Rev Unknown from unknown)]
File  Edit  View  Go  Capture  Analyze  Statistics  Telephony  Tools  Internals  IEEE 802.15.4 Tools  Help
Filter:                                   Expression...
No.   Time         Source     Destination  Protocol       Length  Info
  134 15.493771000 Broadcast               IEEE 802.15.4  14 Fault Injection, Src: Broadcast
  135 15.507735000 0x0000     0x0001       IEEE 802.15.4  19 Data, Dst: 0x0001, Src: 0x0000
  136 15.522716000                         IEEE 802.15.4   5 Ack
  137 15.593753000 0x0000                  IEEE 802.15.4  32 Beacon, Src: 0x0000
  138 15.643732000 0x0000     0x0001       IEEE 802.15.4  19 Data, Dst: 0x0001, Src: 0x0000
  139 15.658738000                         IEEE 802.15.4   5 Ack
  140 15.729772000 0x0000                  IEEE 802.15.4  32 Beacon, Src: 0x0000
  141 15.805754000 0x0000                  IEEE 802.15.4  32 Beacon, Src: 0x0000
  142 15.928738000 0x0000                  IEEE 802.15.4  32 Beacon, Src: 0x0000
  143 16.051735000 0x0000                  IEEE 802.15.4  32 Beacon, Src: 0x0000
  144 16.174716000 0x0000                  IEEE 802.15.4  32 Beacon, Src: 0x0000
  145 16.217770000 0x0000     0x0001       IEEE 802.15.4  19 Data, Dst: 0x0001, Src: 0x0000
  146 16.232735000                         IEEE 802.15.4   5 Ack
  147 16.303747000 0x0000                  IEEE 802.15.4  32 Beacon, Src: 0x0000
  148 16.416731000 0x0000                  IEEE 802.15.4  32 Beacon, Src: 0x0000
  149 16.479725000 0x0000                  IEEE 802.15.4  14 Fault Injection, Src: 0x0000
  150 17.406736000 Broadcast               IEEE 802.15.4  14 Fault Injection, Src: Broadcast
▷ Frame 117: 32 bytes on wire (256 bits), 32 bytes captured (256 bits) on interface 0
▷ IEEE 802.15.4 Beacon, Src: 0x0000
▷ Data (19 bytes)

0000  00 80 6f be ba 00 00 33  cf 00 00 41 74 6d 65 6c   ..o....3 ...Atmel
0010  20 62 65 61 63 6f 6e 20  64 65 6d 6f 20 30 02 5b    beacon  demo 0.[
```

Fig. 1. Tool Architecture Design



		Destination		Source			
Frame Control	Sequence Number	Network Identifier	Address	Network Identifier	Address	Payload	Frame Check Sequence

Type	Security Enabled	Frame Pending	Acknowledgment Requested	Intra Network	Reserved	Destination Addressing Mode	Frame Version	Source Addressing Mode

Fig. 2. IEEE 802.15.4 General Frame Format

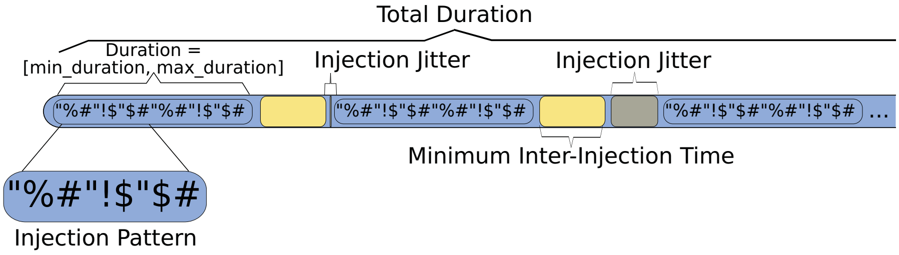


Fig. 3. Graphical representation of some fault-injection parameters

The fault injection unit has two modes of operation: user-defined configurations or pre-configured fault injection profiles.

Both modes originate the fault injection sequence illustrated in Figure 3 accordingly with the parameters that are now explained further.

Fault injection parameters User-defined configuration requires the specification of some parameters in order to build a particular fault injection profile. Some of the parameters are illustrated in Figure 3 and explained further. For future reference, and for the particular case of the IEEE 802.15.4, all time units are to be considered as being in microseconds.

- **Injection Mode -** The injection mode parameter determines how the potentially interfering data will be sent to the wireless transmission medium. When a noise mode is selected, the device simply sends random symbols to the wireless transmission medium until the injection event duration finishes. Secondly, in the preamble preceded mode, the user selected data is attached to the standard preamble and sent to the wireless transmission medium. Finally, in the encapsulated frame mode the user can configure all the fields of a correctly formatted frame.
- **Injection Pattern -** (see Figure 3) the format of the user-defined data, in hexadecimal, to be injected to the wireless transmission medium. If the preamble is used, it will be attached before the indicated pattern; otherwise, only the indicated pattern will be sent. If the defined duration is lower than the time necessary to transmit the pattern this should be trimmed and periodically re-transmitted otherwise.
- **Minimum Duration -** the minimum duration of a single fault injection event. It can be expressed in time units or in bytes;
- **Maximum Duration -** the maximum duration of a single fault injection event. It can be expressed in time units or in bytes. If the minimum and maximum durations are equal, the duration of the fault injection event will always have the same value. Otherwise, it will have a random value between the minimum and maximum durations;
- **Number of Events -** number of total fault injection events in the fault injection campaign, after which the campaign shall terminate.
- **Minimum Inter-Injection Time -** (see Figure 3) the minimum time interval between consecutive fault injection events;

- **Maximum Injection Jitter -** (see Figure 3) maximum value of a positive random time to be added to the Minimum Inter-Injection time.
- **Total Duration -** (see Figure 3) the total duration of a fault injection campaign. If the number of events is specified, the user will no longer be able to set its value, since it will be confined to an lower as well as an upper bound dependant on the specific number of events, its (real) durations, its minimum inter-injection time and its (real) injection jitter durations. Conversely, if the total duration is specified, the fault injection campaign will last until the specified duration is reached. The Total Duration is an alternative to Number of Events parameter described earlier, and for that reason, both parameters cannot be used concurrently;
- **Trigger -** the condition to start the fault injection campaign using the previously described parameters. The trigger can be defined to operate according two modes: one-shot mode, where only one instantiation of the fault injection campaign is performed; and the cyclical mode, where the fault injection campaign is repeated cyclically until the stop command is explicitly issued.

Pre-configured fault-injection profiles Some combinations of these parameters are so relevant that we have decided to include the capability to chose between a set of specific instantiations of the previously described parameters, thus defining pre-configured fault injection profiles. These profiles are useful both for the emulation of accidental faults and for the injection of intentional attacks.

- **Constant -** this profile continuously injects constant noise in the wireless transmission medium. This profile is meant for inducing communication blackouts such as jamming attacks [15];
- **Random -** this profile injects random noise on the wireless transmission medium. This profile is better suited for campaigns where it is only required the occasional corruption of transmissions [15];
- **Adaptive -** This profile tries to synchronize the fault-injection timing with the traffic pattern of the underlying network. Thus the campaign is triggered by network monitoring unit after a specific type of traffic pattern has been detected by the monitoring and capture filter (see Figure 1);
- **Frame-type Adaptive -** specialisation of the **Adaptive** fault injection profile, where faults are injected to destroy the frames matching the specification(s) indicated to the network monitoring unit. This profile is specially useful when, for example, aiming to corrupt only beacon frames causing network-wide blackout [11].

A summary of the relevant fault injection parameters for each of the profiles is represented in Table 1:

3.3 Hardware Integration Interface

The hardware integration interface is a component that enables cooperation and interaction between the network monitoring unit and the fault injection unit, as shown in Figure 1.

	Injection Mode	Minimum Duration	Maximum Duration	Total Duration	Number of Events	Minimum Inter-Injection Time	Injection Jitter	Trigger
Constant	Noise	∞	∞	∞	-	0	0	One-Shot
Random	Noise	Random		-	-	Random	0	One-Shot
Adaptive	Noise	19 bytes	133 bytes	-	1	-	0	Cyclical
Frame-Type Adaptive	Noise	19 bytes	133 bytes	-	1	-	0	Cyclical

Table 1. Fault-Injection Profiles expressed as Fault-Injection Parameters

The first aspect of this interaction concerns the analysis and filtering of the correct received frames. Selected configurations of the Fault-Injector such as pre-configured adaptive and the frame-type adaptive fault injection profiles requires analysis of the captured traffic and in these occasions cooperation between both units is required to trigger a fault injection campaign. In essence any frame field, such as frame-type, addresses and some contents of the payload, can be analysed.

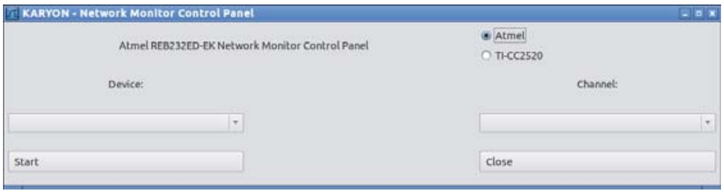
The analysis of the captured traffic allows to detect relevant pre-configured traffic patterns to be used to trigger fault injection campaigns. An effective and highly flexible analysis and filtering of network traffic can be directly mapped into special-purpose hardware components, such as Content Addressable Memories (CAMs), integrated into Field Programmable Gate Arrays (FPGAs) [14].

On the other hand, the start and the end of each fault injection event needs to be issued to the network monitoring unit to be timestamped, ordered, and inserted into the traffic flow to be delivered at the Wireshark Translator (Figure 1).

3.4 Integration with Wireshark

Wireshark is a very flexible reference network analyser. It is extremely general in the sense that it can be extended by the means of plugins to analyse networks and protocols that were not initially considered in its design and engineering. Wireshark can monitor traffic both from real and virtual devices such as networking hardware, regular files and even inter-process communication channels. Beyond that, the captured data can be saved for later analysis. Based on these reasons, Wireshark was chosen to be extended in order to incorporate the control functionalities of the network monitoring and fault injection units.

Firstly, Wireshark communicates directly with the units in order to perform initialisation and command functions. In this sense, an extension the Graphical User Interface was introduced to create a user-friendly interface for units initialization and command purposes. The result of this extension is presented in Figures 4 and 5, respectively for the network monitoring unit and the fault injection unit.



KARYON - Network Monitor Control Panel

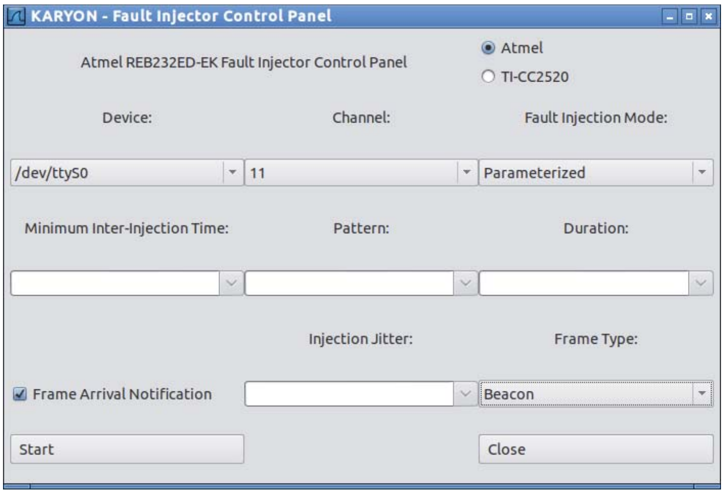
Atmel REB232ED-EK Network Monitor Control Panel

● Atmel
○ TI-CC2520

Device:

Channel:

Fig. 4. Network Monitor Control Panel



KARYON - Fault Injector Control Panel

Atmel REB232ED-EK Fault Injector Control Panel

● Atmel
○ TI-CC2520

Device: /dev/ttyS0

Channel: 11

Fault Injection Mode: Parameterized

Minimum Inter-Injection Time:

Pattern:

Duration:

Injection Jitter:

Frame Type: Beacon

☑ Frame Arrival Notification

Fig. 5. Fault Injector Control Panel

Secondly, after initialization, the network monitoring unit delivers captured frames to Wireshark indirectly that are adapted to the Wireshark format by the Wireshark Translator, as shown in Figure 1.

The Wireshark Translator is a piece of software which has the responsibility to translate the raw frames collected from the network monitoring unit into data understandable by Wireshark. This process requires that a PCAP header [7], the format used by Wireshark, is attached to the raw frame so that Wireshark is aware of, among other aspects, the length and the network type of the frame.

4 Implementation

Built using Commercial Off-The-Shelf (COTS) hardware, our implementation currently supports two hardware boards: the Atmel REB232ED-EK (Figure 6) [3] and the CC2520 and STM32 RFBoards (Figure 7) [13]. Both types of hardware use serial interfaces to connect and exchange data.

The network monitor unit was implemented so it could be easily configured by Wireshark. In Figure 4, it is shown the network monitor control panel. To configure

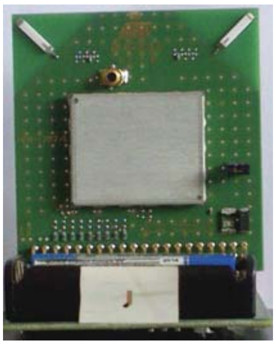


Fig. 6. Atmel REB232ED-EK

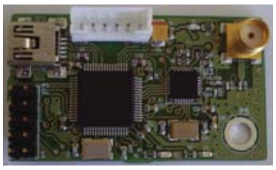


Fig. 7. CC2520 and STM32 RFBoards

a network monitoring session, it is required to select the type and path of supporting hardware (Atmel REB232ED-EK in Figure 6 or CC2520 and STM32 RFBoards in Figure 7) as well as the wireless channel to be monitored.

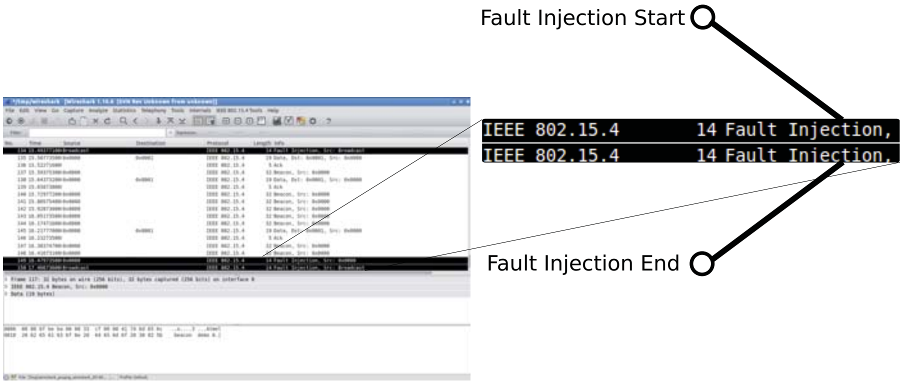
The monitoring starts by signalling the network monitor unit and starting up the Wireshark translator. After a frame capture is successful, the frame length is attached to the frame and sent out via the serial connection as hexadecimal characters.

In this work we aim at providing information making the least amount of assumptions as to who or what is reading the captured traffic and because of that the network monitoring unit represents frames in hexadecimal format. Using hexadecimal format also enables the profiling of IEEE 802.15.4 traffic and error patterns because it allows even simple applications to read the traffic and perform any kind of processing and analysis.

Similarly to the network monitor unit, the fault-injection unit also has a control panel, represented in Figure 5. With this control panel is possible to define fault-injection campaigns making use of the parameters or modes described in section 3.2. This will notify users when a fault injection event begins and when it ends. These delimiter flags will assist the user by clearly stating the frames that arrived during the effects of the fault injection event as shown in Figure 8.

5 Use Cases

This section discusses two relevant use cases: one gives emphasis to the network monitoring functions while the other focuses in fault injection.



```
IEEE 802.15.4        14 Fault Injection,
IEEE 802.15.4        14 Fault Injection,
```

Fig. 8. Wireshark capture screen

5.1 Standalone Network Monitoring

The first use case, illustrates the functionality of our tool in the traffic monitoring of a IEEE 802.15.4 network. However, this network is operating in a ”dirty environment” where a heavy electromagnetic interference is expected from ”alien” wireless nodes.

One of the main sources of interference on the IEEE 802.15.4 channels from wireless ”alien” nodes results from the coexistence with IEEE 802.11 nodes. Since both standards operate in the same 2.4 Ghz Industrial, Scientific and Medical (ISM) bands the probability, in some cases, of both protocols interfere is high [6].

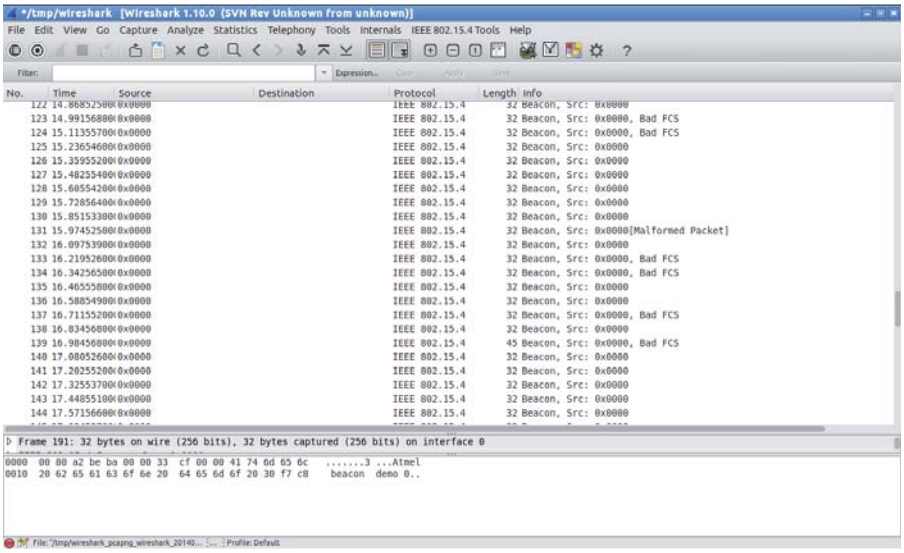
To increase the likelihood of interference, we have set the IEEE 802.15.4 coordinator to channel 17 (2.435 Ghz) which is the closest channel to an ”alien” IEEE 802.11 access point operating in channel 6 (2.437 Ghz). Only by monitoring the beacon frames transmitted by the coordinator, we were able to observe erroneous beacons.

The results from out analysis, inscribed in Figure 9, show a high number of frames disturbed by errors, some of them grouped in bursts of interference. However, these error bursts do not violate (at least in this experiment) the upper bound of three consecutive frame errors defined in the IEEE 802.15.4 standard specification [2].

5.2 Network monitoring with Fault Injection

In this experiment, intended to demonstrate the functionality and effectiveness of the fault injection methodology, the same IEEE 802.15.4 operates in a ”clean environment” where only some occasional frame errors due to the natural electromagnetic interference from the environment are expected. However, a frame injection campaign is conducted, with the following parameters:

- **Injection Mode:** Preamble Preceded Symbols
- **Injection Pattern:** AABBCC
- **Minimum Duration:** 28000 microseconds
- **Maximum Duration:** 28000 microseconds



*/tmp/wireshark [Wireshark 1.10.0 (SVN Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals IEEE 802.15.4 Tools Help
Filter:
Expression...
File: "/tmp/wireshark_pcapng_wireshark_20140... Profile: Default

```
No.   Time          Source    Destination    Protocol        Length Info
  122 14.86852500(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  123 14.99156800(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000, Bad FCS
  124 15.11355700(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000, Bad FCS
  125 15.23654600(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  126 15.35955200(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  127 15.48255400(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  128 15.60554200(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  129 15.72856400(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  130 15.85153300(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  131 15.97452500(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000[Malformed Packet]
  132 16.09753900(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  133 16.21952600(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000, Bad FCS
  134 16.34256500(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000, Bad FCS
  135 16.46555800(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  136 16.58854900(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  137 16.71155200(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000, Bad FCS
  138 16.83456800(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  139 16.98456800(0x0000                     IEEE 802.15.4       45 Beacon, Src: 0x0000, Bad FCS
  140 17.08052600(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  141 17.20255200(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  142 17.32553700(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  143 17.44855100(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000
  144 17.57156600(0x0000                     IEEE 802.15.4       32 Beacon, Src: 0x0000

▷ Frame 191: 32 bytes on wire (256 bits), 32 bytes captured (256 bits) on interface 0

0000  00 00 a2 be ba 00 00 33  cf 00 00 41 74 6d 65 6c   .......3 ...Atmel
0010  20 62 65 61 63 6f 6e 20  64 65 6d 6f 20 30 f7 c8    beacon  demo 0..
```

Fig. 9. Standalone Network Monitoring

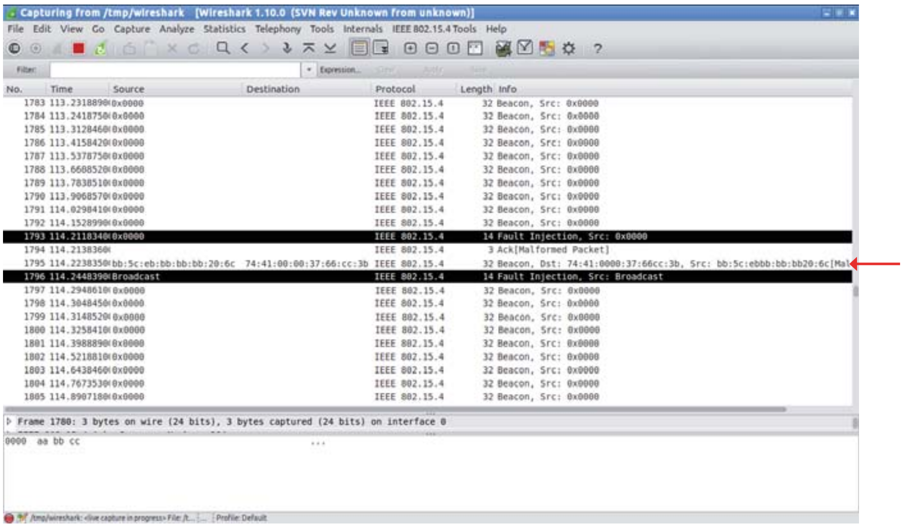
- **Number of Events:** 100
- **Minimum Inter-Injection Time:** 1000000 microseconds
- **Maximum Injection Jitter:** 0 microseconds
- **Total Duration:** Not Defined
- **Trigger:** Cyclic

The results from this fault injection campaign is ilustrated in Figure 10, where the black backgrounded rows signal the begin and the end of a fault injection event. The beacon frame transmitted by the network and signaled by the red arrow at the right of Figure 10 coordinator has been corrupted by the fault injection event. No further frame corruptions have been detected elsewhere in the Wireshark packet list window displayed in Figure 10. It is work noting that in this context the frames designated as ”Ack” in the Wireshark packet list window actually represents the fault injection. The AABBCC pattern when analysed by the Wireshark internal components (in particular, by the wireshark dissector) will be considered as an Acknowledgement frame (see Figure 2).

6 Conclusion and Future Work

In this work we have presented a real-time assessment suite for IEEE 802.15.4 networks. This suite provides two services, namely network monitoring and fault-injection.

Network monitoring allows the capture and analysis of network traffic under normal operating conditions and in the presence of errors. Each captured frame is timestamped with the arrival instant thus supporting the analysis of the real-time characteristics of the network.


Capturing from /tmp/wireshark [Wireshark 1.10.0 (SVN Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals IEEE 802.15.4 Tools Help
Filter:
Expression...
/tmp/wireshark: <live capture in progress> File: /t...
Profile: Default

```
No.   Time        Source                 Destination               Protocol      Length Info
 1783 113.2318890 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1784 113.2418750 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1785 113.3128460 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1786 113.4158420 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1787 113.5378750 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1788 113.6608520 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1789 113.7838510 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1790 113.9068570 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1791 114.0298410 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1792 114.1528990 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1793 114.2118340 0x0000                                           IEEE 802.15.4     14 Fault Injection, Src: 0x0000
 1794 114.2138360                                                  IEEE 802.15.4      3 Ack[Malformed Packet]
 1795 114.2238350 bb:5c:eb:bb:bb:bb:20:6c 74:41:00:00:37:66:cc:3b IEEE 802.15.4     32 Beacon, Dst: 74:41:0000:37:66cc:3b, Src: bb:5c:ebbb:bb:bb20:6c[Mal
 1796 114.2448390 Broadcast                                        IEEE 802.15.4     14 Fault Injection, Src: Broadcast
 1797 114.2948610 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1798 114.3048450 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1799 114.3148520 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1800 114.3258410 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1801 114.3988890 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1802 114.5218810 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1803 114.6438460 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1804 114.7673530 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000
 1805 114.8907180 0x0000                                           IEEE 802.15.4     32 Beacon, Src: 0x0000

▷ Frame 1780: 3 bytes on wire (24 bits), 3 bytes captured (24 bits) on interface 0

0000  aa bb cc                                   ...
```

Fig. 10. Network monitoring with Fault Injection

With this suite effective fault injection campaigns can be easily defined and instantiated while providing means to visualize and record its effects. Such fault injection campaigns are relevant, to test and validate models describing network operations and strategies to improve it.

In particular, both services provided by the suite are useful to study and validate innovative research approaches aiming to bring hard real-time guarantees to WSANs. This has been on of the main motivations for designing and developing this tool.

Future work will include the sophistication of the tool hardware integration interface which in the current version only supports a minimum functionality implemented in software.

References

[1] *AT86RF232 - Low Power, Transceiver for Zigbee, IEEE 802.15.4, 6LoWPAN, RF4CE and ISM Applications*, 2011.

[2] IEEE standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (LR-WPANs). *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pages 1–314, Sept 2011.

[3] *REB232ED-EK - Low Power, Evaluation Kit for Zigbee, IEEE 802.15.4, 6LoWPAN, RF4CE and ISM Applications*, 2011.

[4] Gerald Combs. The Wireshark network protocol analyzer. Available at: `http://www.wireshark.org/`. Accessed in June 5, 2013.

[5] T. Fujiwara, T. Kasami, A. Kitai, and S. Lin. On the undetected error probability for shortened hamming codes. *IEEE Transactions on Communications*, 33(6), June 1985.

[6] I Howitt and J.A Gutierrez. IEEE 802.15.4 low rate - wireless personal area network coexistence issues. In *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, volume 3, pages 1481–1486 vol.3, March 2003.

[7] Van Jacobson and S McCanne. libpcap: Packet capture library. *Lawrence Berkeley Laboratory, Berkeley, CA*, 2009.

[8] A Koubaa, S. Chaudhry, O. Gaddour, R. Chaari, N. Al-Elaiwi, H. Al-Soli, and H. Boujelben. Z-monitor: Monitoring and analyzing IEEE 802.15.4-based wireless sensor networks. In *Local Computer Networks (LCN), 2011 IEEE 36th Conference on*, pages 939–947, Oct 2011.

[9] C.P. O'Flynn. Message denial and alteration on IEEE 802.15.4 low-power radio networks. In *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, pages 1–5, Feb. 2011.

[10] Wolf-Bastian Pöttner and Lars Wolf. IEEE 802.15. 4 packet analysis with wireshark and off-the-shelf hardware.

[11] Jeferson L. R. Souza and José Rufino. Characterization of inaccessibility in wireless networks - a case study on IEEE 802.15.4 standard. In *Analysis, Architectures and Modelling of Embedded Systems. Proceedings of the Third IFIP TC 10 International Embedded Systems Symposium (IESS 2009), Langenargen, Germany, September, 2009. Series: IFIP Advances in Information and Communication Technology.*, September 2009.

[12] Jeferson L. R. Souza and José Rufino. Analysing and reducing network inaccessibility in IEEE 802.15.4 wireless communications. In *38th IEEE Conference on Local Computer Networks (LCN)*, Sydney, Australia, October 2013.

[13] Benjamin Vedder. Homepage of Benjamin Vedder CC2520 and STM32 RF boards. Available at: `http://vedder.se/2013/04/cc2520-and-stm32-rf-boards/`. Accessed in September 25, 2014.

[14] Spartan-3E FPGA family data sheet, August 2009.

[15] Wenyuan Xu, Ke Ma, W. Trappe, and Y. Zhang. Jamming sensor networks: attack and defense strategies. *Network, IEEE*, 20(3):41–47, 2006.

Annex D A Fault-Injection Prototype for Safety Assessment of V2X Communication

Daniel Skarin, Benjamin Vedder, Rolf Johansson, and Henrik Eriksson, “A Fault-Injection Prototype for Safety Assessment of V2X Communication,” in *DEPEND 2014 : The Seventh International Conference on Dependability*, 2014

This page is intentionally left blank.

A Fault-Injection Prototype for Safety Assessment of V2X Communication

Daniel Skarin, Benjamin Vedder, Rolf Johansson, and Henrik Eriksson
Department of Electronics
SP Technical Research Institute of Sweden
Borås, Sweden
e-mail: {daniel.skarin, benjamin.vedder, rolf.johansson, henrik.eriksson}@sp.se

Abstract— **This paper describes an approach for injecting faults in ad hoc vehicle networks. A prototype fault injector, which makes it possible to investigate how a cooperative vehicle system behaves in the presence of communication errors, has been developed. The prototype shows a feasible way to use fault injection as technique to produce evidence for a safety case belonging to a cooperative automotive system.**

Keywords-fault injection, safety assessment, IEEE 802.15.4, V2X communication.

I. Introduction

In the past years, there has been a strong focus on functional safety in the automotive domain. In 2011, the standard ISO 26262 [1] was released, and currently the industry is adopting the development procedure to the standard. At the same time, automotive functions are getting more and more complex; autonomous and cooperative vehicles will soon move from prototypes to products. Safety assessment of cooperative systems will put requirements on evidence which show that communication failures are handled in a safe way. This paper shows a way to inject communication faults in cooperative systems as a technique to produce evidence for a safety case.

Cooperative vehicle systems cover a wide range of interdependence. Willke *et al.* [2] have suggested a taxonomy defining four type levels. On type levels 1 and 2, vehicles and infrastructure are exchanging information without being dependent on it to achieve a safe behavior. On type level 3, the functions rely on communicated information from other vehicles about motion and actuator states to ensure safe and/or efficient operation. On type level 4, applications use inter-vehicle communication to reach a common goal, e.g. driving in a road train (platooning). At least on the type levels 3 and 4, safety requirements will be allocated on the communication between the vehicles (V2V) and between the cars and the infrastructure (V2I).

According to the ISO 26262 standard, safety requirements shall be refined from top-level safety goals to the system components of the physical architecture. For safety-related cooperative functions, this implies that some safety requirements will be put on the V2V and V2I communication, respectively. Furthermore, the standard states what is needed to argue in order to fulfil verification of the safety requirements. For the higher integrity levels (ASIL C and D), it is required to use fault-injection techniques to show that safety mechanisms can handle all safety-relevant faults.

Fault injection in wireless communication used for transfer of safety-critical information in ad hoc vehicle networks needs further research. For computer systems (hardware and software) communicating via wires, there is a fairly long tradition of using fault-injection techniques and tools [3]. Alena *et al*. [4] have investigated how the fault tolerance of wireless sensor networks using IEEE 802.15.4 is affected by interference from other networks and multi-paths. Boano *et al.* [5] present a solution which produce repeatable and precise patterns of interference in wireless sensor networks. Malicious faults (attacks) and some natural faults in ad hoc networks can be assessed using the fault-injection platform developed by de Andrés *et al*. [6].

In this paper, a fault-injection prototype is described. The prototype is based on IEEE 802.15.4 since this standard is used for communication in the automotive and aerospace demonstrators of the KARYON project [7]. However, it is straightforward to adapt the concept to other techniques to be used in the automotive domain (IEEE 802.11p).

Section II introduces relevant fault models originating from functional safety standards. The section also explains how different failure modes can be emulated. Section III describes the fault injection prototype, and Section IV presents initial conclusions and future work.

II. Fault Injection in Communication

A. Fault models

Standards for functional safety, such as ISO 26262 for road vehicles and the generic IEC 61508, list failure modes which are applicable for communication. Part 5 of ISO 26262 [1] lists failure modes for on-chip communication and data transmission. The failure modes for data transmission are applicable for wireless communication. IEC 61508-2 [8] lists identical failure modes for communication. Other important failure modes for communication are blocking access to communication channel [9] and asymmetric information [10]. Table 1 summarizes failure modes applicable for wireless communication.

Based on the diagnostic coverage that is claimed for a safety mechanism, ISO 26262-5 Table D.1 [1] lists failure modes that need to be analyzed. Failure modes for on-chip communication are described next.

Stuck-at failures are described as a continuous low or high signal at the pins of an element. They are applicable for elements which have a pin-level interface for data, control, address, and arbitration signals.

TABLE I. FAILURE MODES FOR COMMUNICATION

Failure mode	Interpretation
Message Corruption	The received data of a message is incorrect.
Message delay	A message is received later than expected by all, or some, receivers.
Message loss	A message is lost by all, or some, receivers.
Unintended message repetition	Receivers obtain two or more messages with the same information instead of one message.
Resequencing	Messages are received with incorrect sequence numbering.
Insertion of message	Receivers obtain a message that they did not expect
Masquerading (or incorrect addressing)	A sender transmit messages using an id of a different sender
Asymmetric information	Information from a single sender is received differently by receivers. It can also be that information from a sender is only received by a subset of the receivers
Blocking access to a communication channel	Prevents nodes from accessing the communication channel, similar to a babbling idiot.

The *direct current* fault model extends stuck-at failures with stuck-open, open, or high impedance outputs, and short circuits between signal lines. The analysis of the fault model is applicable for data, control, address and arbitration signals, but is mainly intended for main signals or on highly coupled interconnections.

When several devices are connected to a bus, arbitration is used to determine which device that controls the bus. *No arbitration* and *continuous arbitration* are mentioned as failure modes for on-chip communication in ISO 26262-5 [1]. *Time out* is mentioned in both IEC 61508 and ISO 26262, but neither standard describes the failure mode in more detail.

Soft errors are caused by ionizing particles, supply voltage noise, or cross-coupling between signal lines. The consequence is one or several bit-flips in memories or bus signals.

B. Emulating the Effects of Faults

The failure modes for wireless data communication can be emulated using a combination of jamming, packet injection, and packet sniffing. Jamming [5][11] is used to prevent one or several nodes from receiving or sending packets. Packet injection is used to insert additional, duplicated or corrupted messages in the wireless network. Packet sniffing allows the fault injection module to eavesdrop the wireless traffic in a non-intrusive manner. This is useful for logging and for triggering the injection of different failure modes.

Table 2 shows how different failure modes can be implemented by combining jamming and packet injection. For example, the effects of a message delay can be emulated by jamming to prevent nodes from receiving the original message, and then resending the original message with a delay. This assumes that we have a priori knowledge of the content of the message. Message losses are emulated by activating jamming when specific messages are being transmitted by a node.

Figure 1 and Figure 2 illustrate how failure modes for on-chip communication are emulated. The signal between two elements passes through a fault injection module which has the capability to modify the transmitted signal value. For most failure modes, such as soft errors, a faulty signal only relies on the value of the non-faulty signal as shown in Figure 1. For short-circuits between signals, however, the values of two or more signals are needed, as shown in Figure 2.

TABLE II. EMULATING FAILURE MODES USING JAMMING AND PACKET INJECTION

Failure mode	Jamming	Packet Injection
Message Corruption	x	x
Message delay	x	x
Message loss	x	
Unintended message repetition		x
Resequencing		x
Insertion of message		x
Masquerading (or incorrect addressing)		x
Asymmetric information	x	x
Blocking access to a communication channel	x	

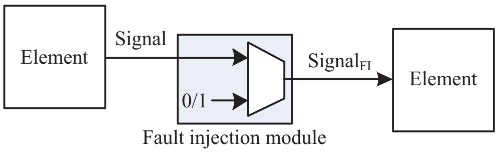


Figure 1. Injection of stuck-at faults in a signal.

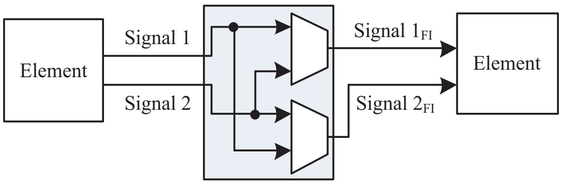


Figure 2. Injection of short-circuit failures between two signals.

C. Controlling When to Inject Faults

Figure 1 shows a state machine for controlling the fault injection. The idle state has an internal counter to keep track of the currently evaluated trigger. When all triggers have been evaluated to true in the correct order, fault injection is activated in the state “Start FI”. Following that, the “FI” state is immediately entered.

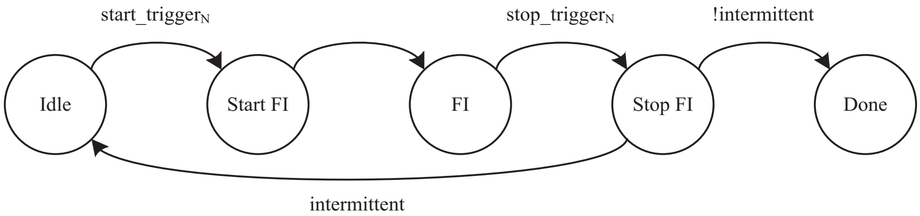


Figure 3. State machine to control the fault injection.

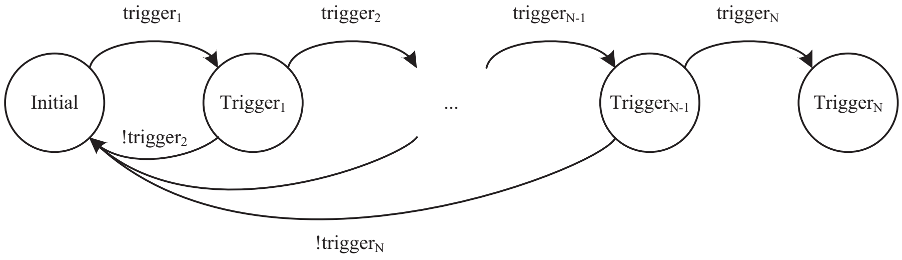


Figure 4. State machine to handle start and stop triggers for fault injection.

The "FI" state is exited when all stop triggers have been fulfilled. Unless an intermittent fault is emulated, the fault injection is stopped. For intermittent faults, there is a return to the idle state and another wait for start trigger fulfillment. Fault injection is activated using triggers which can be based on: elapsed time, probability per received packet, sender or receiver address of a packet, or data in the payload of a packet. Several triggers can be combined so that fault injection is started or stopped by a chain of events, as shown in Figure 2. Using this approach, well-known packet loss models such as Bernoulli and Gilbert-Elliot [12] can be supported, as well as simple triggers based on, e.g., elapsed time.

III. FAULT INJECTION PROTOTYPE

The fault injection concept described in the previous section has been implemented for vehicle demonstrators in the KARYON project [7]. The fault injection prototype can be used for injecting failures in IEEE 802.15.4 data communication, and in the on-chip communication. Figure 5 shows a picture of the fault injection node, which uses the STM32F4 microcontroller from ST and the CC2520 communication chip from Texas Instruments. The node is based on layout and hardware schematics which are freely available from [13].

The fault injector uses ChibiOS/RT [14] as its operating system, and implements the state machine described in Section II.C. The following fault injection triggers are supported:

- Time – Enabled after a specified time has elapsed.
- Packet probability – Enabled with a specified probability for each received packet.
- Packet destination address – Enabled when a packet with a matching source address is received.
- Packet source address – Enabled when a packet with a matching destination address is received
- Packet data – Enabled when the specified data matches the received data
- The fault injector is configured using USB commands, or by sending configuration packets via IEEE 802.15.4.

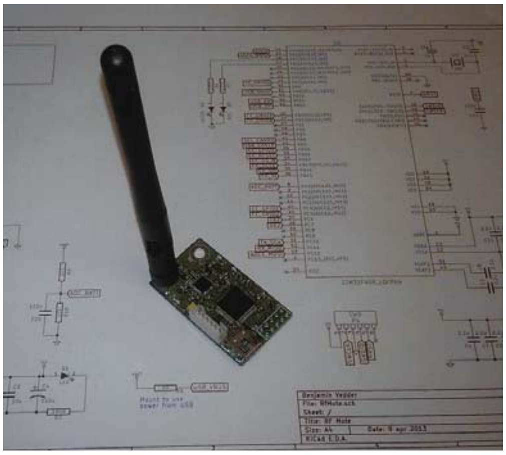
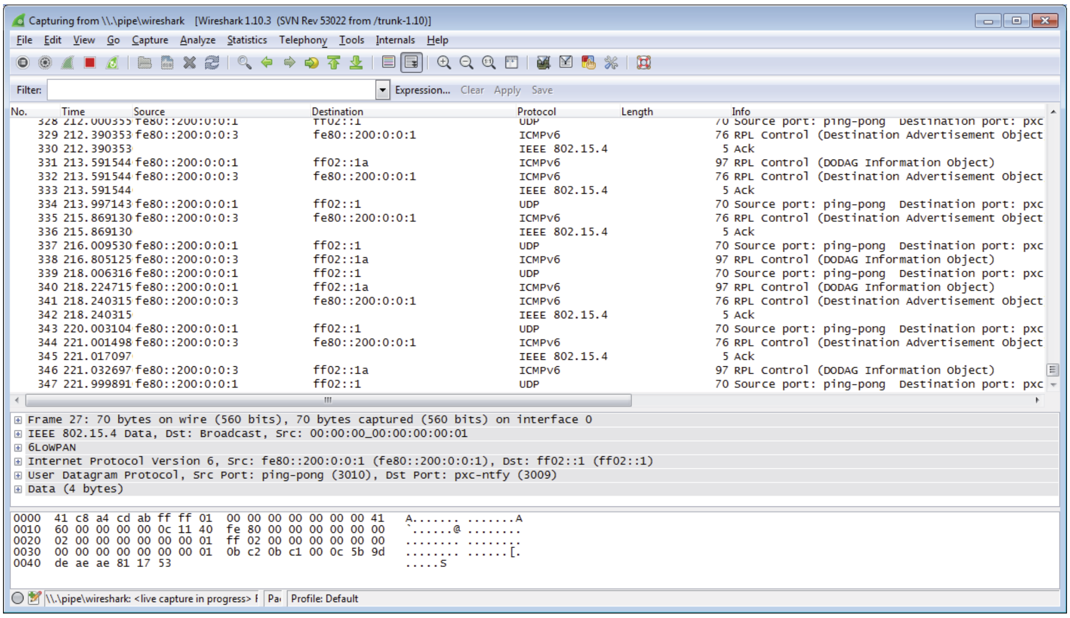


Figure 5. RF board with STM32F4 and CC2520 based on [Vedder].

The prototype fault injector also provides packet logging capabilities, which are useful for debugging purposes. The CC2520 communication chip provides hardware support for packet sniffing, which can be used as a non-intrusive method of observing wireless traffic. The fault injector can output captured packets in the packet capture (pcap) format using a named pipe. The logged traffic can then be analyzed in real-time using tools such as Wireshark which is an open source network protocol analyzer. Figure 6 shows an example of logged traffic in Wireshark.

The following failure modes are currently supported by the fault injection prototype: message corruption, delay, loss, insertion, unintended message repetition, masquerading, and blocking access.


Capturing from \\.\pipe\wireshark [Wireshark 1.10.3 (SVN Rev 53022 from /trunk-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.   Time       Source            Destination       Protocol       Length  Info
  328 212.000355 fe80::200:0:0:1   ff02::1           UDP                    70 Source port: ping-pong  Destination port: pxc
  329 212.390353 fe80::200:0:0:3   fe80::200:0:0:1   ICMPv6                 76 RPL Control (Destination Advertisement Object
  330 212.390353                                     IEEE 802.15.4           5 Ack
  331 213.591544 fe80::200:0:0:1   ff02::1a          ICMPv6                 97 RPL Control (DODAG Information Object)
  332 213.591544 fe80::200:0:0:3   fe80::200:0:0:1   ICMPv6                 76 RPL Control (Destination Advertisement Object
  333 213.591544                                     IEEE 802.15.4           5 Ack
  334 213.997143 fe80::200:0:0:1   ff02::1           UDP                    70 Source port: ping-pong  Destination port: pxc
  335 215.869130 fe80::200:0:0:3   fe80::200:0:0:1   ICMPv6                 76 RPL Control (Destination Advertisement Object
  336 215.869130                                     IEEE 802.15.4           5 Ack
  337 216.009530 fe80::200:0:0:1   ff02::1           UDP                    70 Source port: ping-pong  Destination port: pxc
  338 216.805125 fe80::200:0:0:3   ff02::1a          ICMPv6                 97 RPL Control (DODAG Information Object)
  339 218.006316 fe80::200:0:0:1   ff02::1           UDP                    70 Source port: ping-pong  Destination port: pxc
  340 218.224715 fe80::200:0:0:1   ff02::1a          ICMPv6                 97 RPL Control (DODAG Information Object)
  341 218.240315 fe80::200:0:0:3   fe80::200:0:0:1   ICMPv6                 76 RPL Control (Destination Advertisement Object
  342 218.240315                                     IEEE 802.15.4           5 Ack
  343 220.003104 fe80::200:0:0:1   ff02::1           UDP                    70 Source port: ping-pong  Destination port: pxc
  344 221.001498 fe80::200:0:0:3   fe80::200:0:0:1   ICMPv6                 76 RPL Control (Destination Advertisement Object
  345 221.017097                                     IEEE 802.15.4           5 Ack
  346 221.032697 fe80::200:0:0:3   ff02::1a          ICMPv6                 97 RPL Control (DODAG Information Object)
  347 221.999891 fe80::200:0:0:1   ff02::1           UDP                    70 Source port: ping-pong  Destination port: pxc

⊞ Frame 27: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0
⊞ IEEE 802.15.4 Data, Dst: Broadcast, Src: 00:00:00_00:00:00:00:01
⊞ 6LOWPAN
⊞ Internet Protocol Version 6, Src: fe80::200:0:0:1 (fe80::200:0:0:1), Dst: ff02::1 (ff02::1)
⊞ User Datagram Protocol, Src Port: ping-pong (3010), Dst Port: pxc-ntfy (3009)
⊞ Data (4 bytes)

0000  41 c8 a4 cd ab ff ff 01  00 00 00 00 00 00 00 41   A....... .......A
0010  60 00 00 00 00 0c 11 40  fe 80 00 00 00 00 00 00   `......@ ........
0020  02 00 00 00 00 00 00 01  ff 02 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 01  0b c2 0b c1 00 0c 5b 9d   ........ ......[.
0040  de ae ae 81 17 53                                  .....S
```

Figure 6. Packet sniffing using the fault injection node and Wireshark.

For some of the failure modes, e.g. message delay, message payload need to be known a priori. Proof-of-concept fault injections have been successfully performed, but no complete fault-injection campaigns have been run yet.

IV. Conclusions and Future Work

A prototype fault injector for digital communication, in particular wireless communication, has been described. One limitation with the approach is that communication chips require some time to switch between receiving and sending. For the CC2520 chip, the RX/TX turnaround time is 192µs. For packets with a small payload, it might therefore not be possible to trigger the fault injection and jam the packet currently being received. This is something which will be investigated in the near future.

The prototype has been tested on IEEE 802.15.4 communication, but the concept is straightforward to adapt to other communication techniques, such as IEEE 802.11p.

The prototype shows that it is feasible to inject most faults needed in a safety assessment according to the requirements in functional safety standards.

Acknowledgment

This work has been supported by the EU under the FP7-ICT program, through project 288195 Kernel-based ARchitecture for safetY-critical cONtrol (KARYON).

References

[1] ISO26262-5, “Road vehicles – Functional safety – Part 5: Product development at the hardware level”, 2011.

[2] T. L. Willke, P. Tientrakool, and N. F. Maxemchuk, “A survey of inter-vehicle communication protocols and their applications”, IEEE Communications Surveys & Tutorials, vol. 11(2) , pp. 3-20, 2009.

[3] H. Mei-Chen, T. K.Tsai, and R. K. Iyer, “Fault injection techniques and tools”, IEEE Computer, vol. 30(4), pp. 75-82, 1997.

[4] R. Alena, R. Gilstrap, J. Baldwin, T. Stone, and P. Wilson, “Fault tolerance in ZigBee wireless sensor networks”, Proc. 2011 IEEE Aerospace Conference, March 2011, pp. 1-15.

[5] C. A. Boano et al., “Controllable radio interference for experimental and testing purposes in wireless sensor networks,” Proc. IEEE 34th Conference on Local Computer Networks, Oct. 2009, pp. 865-872.

[6] D. de Andrés, J. Friginal, J.-C. Ruiz, and P. Gil, ”An attack injection approach to evaluate the robustness of ad hoc networks”, Proc. 15th IEEE Pacific Rim International Symposium on Dependable Computing, Nov. 2009, pp. 228-233.

[7] Homepage of Kernel-Based ARchitecture for safetY-critical cONtrol, http://www.karyon-project.eu/, accessed on 25th of June 2014.

[8] IEC 61508-2, “Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems”, 2010.

[9] H. Kopetz, “Real-time systems”, Kluwer, 1997.

[10] F. Cristian, “Understanding fault-tolerant distributed systems”, Communications of the ACM, vol. 34, pp. 56-78, 1991.

[11] A. D. Wood, J. A. Stankovic, and G. Zhou, "DEEJAM: Defeating energy-efficient jamming in IEEE 802.15. 4-based wireless networks," Proc. 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON'07), June 2007, pp. 60-69.

[12] J.-P. Ebert and A. Willig, “A Gilbert-Elliot bit error model and the efficient use in packet level simulation”, Technical Report, TKN-99-002, Technical University of Berlin, 1999.

[13] Homepage of B. Vedder “CC2520 and STM32 RF boards”, http://vedder.se/2013/04/cc2520-and-stm32-rf-boards/, accessed on 25th of June 2014.

[14] Homepage of ChibiOS/RT, http://www.chibios.org, accessed on 25th of June 2014.

Annex E A Failure Injection Framework for Cyber-Physical Systems

S. Zug, T. Brade, C. Steup, “A Failure Injection Framework for Cyber-Physical Systems”, White Paper EOS-OVGU

This page is intentionally left blank.

A Failure Injection Framework for Cyber-Physical Systems

Sebastian Zug, Tino Brade, Christoph Steup
Otto-von-Guericke University Magdeburg, Germany
{zug, brade, steup}@ovgu.de

ABSTRACT

Cyber-Physical Systems (CPSs) control real-world entities and inherently depend on the safety of operation. Standards like ISO 26262 or ISO 61508 lay down rules and procedures for fault injection techniques to verify the reliability of CPSs in the automotive and industrial automation field respectively. A system tolerating a larger number of injected faults or a higher occurrence rate usually imply a higher reliability and, in turn, may increase safety. One of the core issues is defining the fault model which anticipates the nature of faults or fault patterns, the amplitude the occurrence probability and distribution. This is a prerequisite for a successful fault injection analysis. These fault assumptions have to match exactly the individual safety goals of an application. Overdoing, i.e. assuming non-relevant fault pattern, amplitudes and occurrence probabilities will increase costs and probably development time without a measurable benefit for safety. On the other hand, too optimistic assumptions will decrease the needed safety level. Thus, the set of injected faults must precisely match the safety requirements of a CPS.

We propose a versatile and flexible fault injection framework for CPS that particularly considers sensor failures. To achieve a high level of flexibility and adaptability to various safety requirements, we capture the properties of relevant faults defined by the standards as well as the faults emitted by the sensors in an Electronic Data Sheet (EDS). A fault type and its type specific parameters describe the characteristics of individual faults or more complex fault patterns. Our framework is based on Simulink. A parser evaluates the EDS and includes this data into pre-defined fault injection blocks. This integrates fault injection early in the design phase. The overall schedule for fault injection containing all relevant faults is automatically derived from the descriptions to satisfy the needs of the respective standard. Additionally, we use the sensor fault information to exactly include those faults only that are relevant for the safety goals of the application. This allows analyzing and verifying the needed safety requirements in early stages of the design, keeping the necessary overhead to a minimum.

Categories and Subject Descriptors

B.2.3 [**Arithmetic and Logic Structures**]: Reliability, Testing, and Fault-Tolerance—*Test generation*; D.2.8 [**Software Engineering**]: Requirements/Specifications—*Languages*

Keywords

Fault injection, testing, safety, sensor faults, Simulink

1. MOTIVATION

"Cyber-Physical Systems" [1], "autonomous automotive systems", "smart environments" [2] or "Pervasive/Ubiquitous Computing" [3] intimately link the entities of the real-world to their virtual images in (a set of) computers. At the interface between these worlds, sensors perceive and transform the real-world entities. The challenge is to provide a precise and an accurate observation that is substantially affected by sensor failures and inherent uncertainties. Obviously, these failures cause a potential risk for violating safety goals, because an application takes action based on these observations. A safety critical system must be proven safe before it will be put to operation due to the potential to cause harm. Therefore, checking, analysis and validation have to be performed at early design and development stages. Failure injection is one of the important techniques providing evidence at design-time that the application is either robust enough to resist failures or not susceptible to the considered set of failures. Recognizing the advantages, failure injection became part of several standard such as ISO 26262.

ISO 26262 defines automotive safety integrity levels (ASIL) that identifies the safety requirements for a function that may be broken down to requirements of components that are involved. The respective ASIL is derived from a Hazard and Risc Analysis (HARA) and defines the requirements of safety mechanisms. In order to verify such mechanisms, test patterns are mandatorily demanded to be injected. The characteristics of such test patters depends on the safety integrity attribute (ISO 26262 - ASIL A-D, ISO 61508 - SIL 1-4) that is derived from HARA. For instance, out-of-range, offsets, stuck in range and oscillations faults are specified by the standard. However, standards only specify single-point faults and fail to address complex fault patters, which are essential in order to reproduce the behavior of a sensor. In addition, the selection of the injection points (e.g. hardware or software) as well as the parameterization of the injected faults is left to the system engineer.

It should be noted that the generation of relevant, complex failure patterns is by no means easy or straightforward. As indicated above, too pessimistic assumptions about the failure class, occurrence probability and distribution may lead to an overly expensive design while, even more dangerous, too optimistic settings may lead to the situation where the system may be erroneously compliant with a higher ASIL. This may jeopardize safety at run-time. Especially, the latter situation must be avoided under all circumstances.

The proposed failure injection framework provides a compliance check and a failure injection monitor in order to generate complex test patterns. The compliance check identifies the failure types such that the requirements of the safety goals are met. To know how often those failures need to be injected, the failure injection framework features a monitoring functionality to obtain how many failures have been injected so far. Further information on this topic is provided by the following section. The third section refers to the state of the art. The final section of the paper summarizes our achievements and mentions the future work.

2. FAILURE INJECTION FRAMEWORK

This section presents a failure injection framework for verifying the robustness of an application by injecting sensor failures. We are talking about failure injection because we modify the output data of a sensor (thus generating a failure) rather than injecting faults in the internal components of a sensor. The rational behind this approach is that we may deal with sensor data generated by a simulator during the design phase or a sensor in a lab environment respectively. In both cases, we would obtain perfect or near perfect sensor readings. By applying failure injection, the difference between a real run-time behavior of a certain sensor and the design-time (simulation- or lab-based) analysis of an application will be compensated. A real physical sensor reading may suffer from multiple uncertainties due to internal faults, sensor characteristics, signal conditioning, EMC, changes in the environment or non-steadiness of the observed process. Because of these failures, an application may take wrong actions and safety constraints may be violated. Therefore, we have to consider defining sophisticated failure modes for our injection tool.

For representing complex failure cases, several failure injection units are essential, which do not provide a result that matches with real observations. When failure injection units are running on their own, some failure combinations might not be injected whereas other combinations are injected which do never occur when using a sensor in reality. Therefore, failure injection of complex patterns requires a kind of schedule so that each failure injection instance is activated at the right time. The generation of such a schedule brings us finally to the questions: what failure types need to be injected and how many of them for being compliant with a standard.

Table 1: Comparison of failure types presented in the EDS and defined by the standard

		electronic sensor data sheet	
		present	*not* present
ISO 26262	present	mandatory	critical
	not present	unnecessary	not relevant

As mentioned in the motivation, there is a chance to inject either too many failures or not enough failures. As shown in the Tab. 1, a failure does not need to be considered, which is neither required to be injected by the standard nor exhibited by a sensor. When failures of the standard match with the failure behavior of a sensor, a mandatory failure injection configuration has been found. Requested failures of the standard are considered to be unnecessary if a sensor does not exhibit those failures. Appearing failures of a sensor are stated to be critical in case the standard does not consider them. Considering the second question "how many failures have to be injected", a failure injection monitor is required beside the failure injection schedule. When no enough failures have been injected, the system seems sound but is in fact not as robust as required. Of course, this situation needs to be avoided by all efforts because it features a safety attribute that cannot be maintained. On the other hand, injecting more failures than required leads to an overly robust design, which is in principle acceptable but at the expense of system performance and system complexity.

In order to master those questions, the proposed failure injection framework operates on a failure description that is linked to a failure model. This allows describing in separate the individual failure behavior of a sensor as well as the required failures to be injected of a standard. The key for matching the failures of a sensor with a standard is the failure model, which defines explicitly the failure types provided by the failure injection framework. Afterwards, the failure description is used to generate a schedule that matches the failure behavior of a real sensor. The generated schedule serves as a configuration for the failure injection framework. Finally, the injected failures are monitored at run-time in order to master the trade-off between injecting not enough failures and injecting too many failures.

2.1 Definition of the Failure Model

We defined a set of failures types in [4], which we found are relevant for sensors. Partly, these failure types have been described in the literature [5], partly they have been observed in experiments, which we conducted in our lab. The failure model comprises timing failures and value failures. Referring to the analysis in [4], e.g. delay and omission failures belong to timing failures. The class of outlier failures, spikes, offsets, drift, noise, stuck-at-X and saturation failures belong to value failures. The survey in table Tab. 2 distinguishes 13 failures. Computational or network failures are currently not part of the model. However, it should be noticed that our failure injection framework is not limited to the mentioned failure types and may include other failures in future work. Thus, we designed the framework in a generic and flexible way that enables further individual failure types to be included.

Starting from the defined failure types we need to specify the way in which they are generated. We may need to activate a failure with reproducible characteristics many times.

Table 2: Characterizing modelled failures

I	II	III	
Failure type	Occurrence	Parameters	
		Amplitude	Corr.
Constant Noise	transient	pdf	
Time-Corr. Noise	transient	pdf	function
Value-Corr. Noise	transient	pdf	function
Constant Offset	permanent	value	
Time-Corr. Offset	permanent	value	function
Value-Corr. Offset	permanent	value	function
Outlier	transient	pdf	
Spike	transient	value	
Stuck-At-X	permanent	pdf	
Stuck-At-Zero	permanent		
Saturation	permanent	value	
Constant Delay	permanent	time	
Time-Corr. Delay	permanent	time	function

Therefore we can specify an occurrence probability in addition to the failure type and amplitude. The failure amplitude is determined by a probability density function over time. In cases where a non-linear behavior is required, an individual function is used to describe the failure amplitude, which allows reproducing complex signal waveforms such as spikes. On the other hand, the occurrence of a failure describes the point in time when a particular failure is going to be injected. A failure type specified as permanent will be injected in every simulation step. In contrast, transient failures will occur stochastically defined according to the occurrence probability.

2.2 Failure Description

From a careful analysis of a sensor, that has been analytical and empirical, we were able to identify which sensor failures may occur and what are their characteristics. This knowledge, captured in a detailed failure description is used to configure the failure injection framework. For achieving compatibility between systems, the failure description is encoded in a compact Extended Markup Language (XML) data structure. Listing 1 shows the failure behavior of a sensor where each failure type is described together with its respective parameters. This failure description holds the parameters of constant noise (from line 3 to line 8) and outlier failures (from 9 to line 14) where the occurrence probability, the used distribution and the correlation are described respectively.

To describe more complex sequences of failures, we also need to model the transition probabilities between various failure states including the failure-free case. We specify this behavior by a Markov model, as depicted in Fig. 1, in which the transition matrix describes the stochastic behavior of the failures. Such a transition matrix holds the probability of staying in the same state (self-loop) and of switching to another state where n states represent the n failure types and an additional state corresponds to the failure-free behavior. The description of a transfer matrix is given in Listing 1 from line 15 to line 19, which is interpreted as follows. Assuming the sensor being in the failure-free state, which is represented by line 16, the sensor stays with a probability of 92 per cent in the failure-free state (self-loop), switches with a probability of 7 per cent to the noise state and enters the outlier state with a probability of 1 per cent. By integrating the transition matrix in the injection concept, we are able to uniquely describe complex failure behavior.

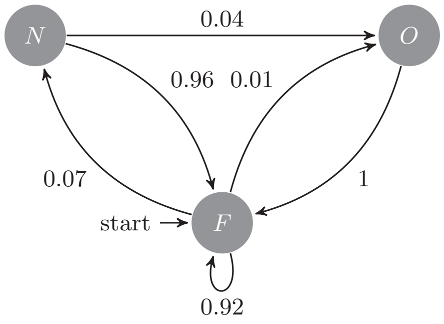


Figure 1: Markov model representing the failure behavior of a sensor

Listing 1: Sensor-specific failure description

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <failure_types>
3    <failure_N>
4      <type>value correlated noise</type>
5      <occurrence>0.1</occurrence>
6      <distribution_id>1</distribution_id>
7      <correlation_id>0</correlation_id>
8    </failure_N>
9    <failure_O>
10     <type>outlier</type>
11     <occurrence>0.01</occurrence>
12     <distribution_id>2</distribution_id>
13     <correlation_id>0</correlation_id>
14   </failure_O>
15   <transitions>
16         <F>0.92;0.07;0.01</F>
17         <N>0.96;0;0.04</N>
18         <O>1;0;0</O>
19   </transitions>
20 </failure_types>
```

By applying the same description format to a safety standard, the compliance check as well as the monitoring can be performed automatically. Therefore, the application-specific description holds failure types, which are essential for verifying the robustness of an application. The outcome of the compliance check classifies the described failure types in accordance to Tab. 1. On the other hand, the probabilities of an application-specific description specify how precise the considered failure types have to be injected. This aspect is addressed by the failure monitor, introduced in Sec. 2.5.

2.3 Generating the Failure Schedule

The failure schedule is key for running failure injection of complex patterns in a deterministic and repeatable manner. First, the set of failures need to be identified that have to be injected. Second, activation signal have to be generated

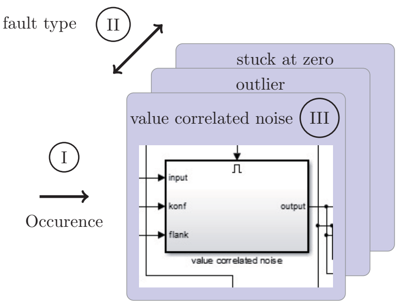


Figure 2: Failure injection scheme

in accordance to the identified failures. Third, the schedule have to be assigned.

The failures to be injected result from checking the failure description of the sensor against the application-specific failure description. This allows us to classify failures as critical, mandatory, unnecessary or not even relevant, as shown in Tab. 1. For achieving this, the failure types of the sensor as well as the application are mapped as terms of regular expressions. At this point, the compliance check is performed by matching the regular expressions against each other. In case both expressions match, mandatory failures have been found. When a sensor expression do not match with the standard, the regular expression represents a critical failure to be checked. Failures are stated unnecessary if regular expression of the standard do not match the expression of the sensor. Otherwise, failures are considered to be not relevant and can be omitted. The generation of a schedule fails if a sensor datasheet contains a failure type, which does not fit any specified type in the standards definition (application-specific failure description). In this case, the developer has to check the requirements deduced by the HARA or has to replace the sensor. Using regular expressions representing failure types turn out to be beneficial for injecting the right failures and omitting failures that are not of importance.

When the compliance check have been successfully passed, activation signals are generated for failures that are classified as mandatory and critical. Those activation signals are used to trigger the corresponding implementation of a specific failure type. Consequently, failure types can be individually activated for injecting complex failure patterns. The same activation signal results in the same injected failure pattern over and over again, which makes this approach predestinated for verifying the robustness of an application at design-time as well as providing evidence that safety constraints are met.

Finally, the failure injection schedule is a result of linking those activation signals with the Markov model, which is configured by the transition matrix given by the failure description of the selected sensor. The Markov model is used to calculate the probability of the following failure state. When linking failure states with activation signals, the failure injection follows the statistical nature of the described transition matrix. The described failure injection schema is depicted in Fig. 2.

2.4 Performing the Failure Injection

The presented failure injection framework is implemented in MATHWORKS Simulink [6] and consist of three functional blocks: Failure Control Unit (FCU); Failure Schedule Block (FSB) and Failure Injection Block (FIB).

The FCU reads in the failure descriptions, parses them into regular expressions and finally checks the compliance between sensor and the required standard, as described in Sec. 2.2. When passing the compliance check, a set of activation signals is generated for failures that are stated to be mandatory as well as critical (to be checked). Such activation signals trigger Simulink blocks where each block implements the failure injection of a specific failure type. Afterwards, the FSB selects a specific activation signal based on the Markov model given by the failure description. Finally, the FIB generates the output of the failure injection. There may be failures that just distort the original sensor signal. In these cases, the output signal is obtained by combining, e.g. adding or multiplying the failure pattern and the original sensor signal. This is e.g. applied to inject for instance offset, noise and outlier failures. Other failures are completely independent of any sensor signal as e.g. as stuck-at-zero or saturation failures. In these cases, the injector just replaces the original sensor data with the specific failure pattern.

Fig. 3 illustrates the outcome of the failure injection framework and shows the injection of a constant noise failure. The second example displayed in Fig. 4 shows the injection of constant noise and outlier failures for the same signal pattern.

2.5 Monitoring Injected Failures

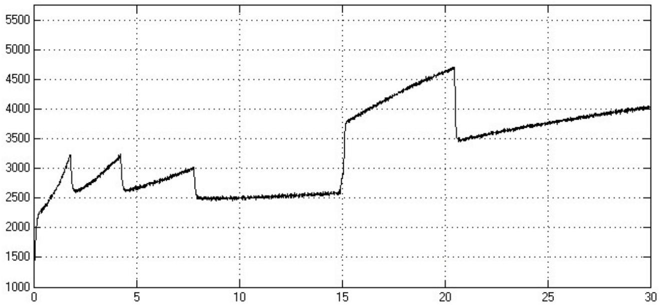


Figure 3: sensor data with injected constant noise

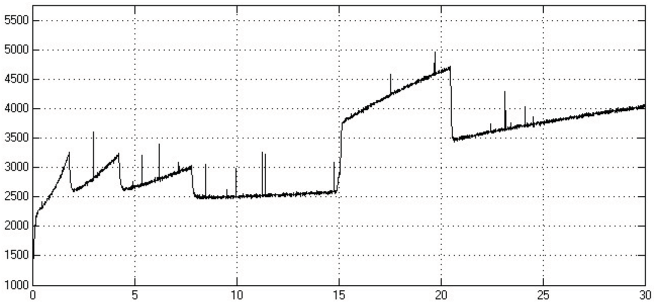


Figure 4: sensor data with injected constant noise and outliers

The monitoring of injected failures is used to master the tradeoff between injecting not enough failures and injecting too many failures. In order to provide an answer to the question how many failures have been injected so far and still need to be injected, a second Markov model is used to trace activation signals. Whenever an activation signal is triggered, the corresponding failure state will be counted. The key for doing this is a bijective function between activation signals and the states of the Markov model. While generating the schedule, states are mapped to activation signals. Whereas to provide the monitoring functionality, the other way around is needed.

As a consequence, we receive Markov models describing the failure behavior of both, the real sensor and the set of injected failures. By comparing them, we can check whether our assumptions about the occurrence rate match reality. Each Markov model is described by a respective transition matrix. The failure behavior of the failure injection perfectly matches the real sensor if the difference of both matrices equals zero, as shown in the first row of the results matrix in Fig. 5. In the case that the probabilities of the transition matrix of the failure injector are greater than those of the transition matrix of the real sensor, then we injected too many failures, as depicted in the second row of the results matrix. Otherwise, the number of injected failures may be too low, as depicted in the last row of the results matrix.

$$\overset{\text{Sensor}}{\begin{array}{c|cccc|} & f_1 & f_2 & \cdots & f_n \\ f_1 & p_{11} & p_{12} & \cdots & p_{1n} \\ f_2 & p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_n & p_{n1} & p_{n2} & \cdots & p_{nn} \end{array}} - \overset{\text{Injection}}{\begin{array}{c|cccc|} & f_1 & f_2 & \cdots & f_n \\ f_1 & p_{11} & p_{12} & \cdots & p_{1n} \\ f_2 & p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_n & p_{n1} & p_{n2} & \cdots & p_{nn} \end{array}} = \overset{\text{Results}}{\begin{array}{c|cccc|} & f_1 & f_2 & \cdots & f_n \\ f_1 & 0 & 0 & \ldots & 0 \\ f_2 & -0.2 & -0.1 & \ldots & -0.3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_n & 0.1 & 0.3 & \ldots & 0.15 \end{array}}$$

Figure 5: Transition Matrix

3. STATE OF THE ART

Fault injection techniques can be applied on different stages in an ISO 26262 development process - 5-10 ”Hardware integration testing“, 6-10 ”Software integration testing“, 4-9 ”Safety validation“, etc. To cover this spectrum, we have to consider different approaches for fault injection. A comprehensive overview is presented for instance in [7]. We derived three categories to summarize the state of the art:

3.1 Way of injection

An overview of fault injection techniques is given in [8]. We can divide three basic concepts - hardware, software and simulation based approaches. The first one directly effect a component by electrical signals to manipulate pins or memory elements [9]. Contactless intrusions uses electrical fields and radiation to initiate disturbances [10]. Software-based fault injection components trigger a fault on the embedded target system. Simulation-based fault injection can be accomplished in at different levels of abstraction e.g. logical levels or system level [9].

3.2 Definition of fault models

The intended fault models are strongly connected to the way of injection [8]. Hardware based fault injection considers digital fault models - stuck-at or flipped bits. Measurement oriented injection techniques apply continuous fault models like noise, outliers or spikes [11]. Our data centric and sensor specific fault type classification was motivated by different previous work, for instance [5], [7]. They combine 14 basic fault models suitable for a large number of sensing devices (distances, temperature, orientation, acceleration, etc.).

3.3 Selection of relevant faults

The generation of a minimal test plan is in the focus of many research papers. Often a simulation is used to recognize the relevant faults in a first step. The authors of [12] tried to reduce the amount of test cases through the complete analysis of different fault sets. After the simulation the test schedule is executed on a real target.

3.4 Schedule declaration

The injection schedule contains a fine grained activation plan of relevant faults. A common approach is a definition is safety oriented frameworks like SCADE or Simulink. A multi domain approach is discussed in [13]. In all cases the developer has to “implement” faults and their order manually. The authors of [14] refined the approach and developed a new language to specify fault injection patterns, but the fault models are limited to discrete fault states.

The approach described in this paper is focused on simulation based fault injection. Similar approaches in mentioned papers do not close the gap between standard requirements and fault characteristic of the used components. Bozzano, Villafiorita, Åkerlund, *et al.* illustrate this fact in a compact way when they depict a safety engineer and a design engineer in their architecture. Both developers are responsible for requirement capturing but just the safety engineer defines the relevant faults. But decisions about used components are done by the design engineer. Just he can know their fault characteristic. An efficient and seamless fault injection tool-chain is not possible in such a development process.

4. CONCLUSION AND FUTURE WORK

We proposed a failure injection framework respecting safety goals of an application. Unlike traditional failure injection techniques, we recommend using a failure model to balance the trade-off between injecting too many failures and not enough failures, instead of injecting failures without reference to the source of failures as it is common. In order to achieve that, we first compute the probability of a certain failure and in a second step we generate the related amplitude of the failure to be injected. This allows at design-time to inject precisely the set of failures from which a CPS may suffer at run-time. The respective safety goal of the CPS defines the degree of similarity which the injected failures must have compared to the real failure behavior. In this paper, we presented a way to meet the compliance with the occurrence rate.

The exact verification of the amplitude of a failure and the defined failure types needs further investigations. Matching the failure model with a realistic failure behavior is of utmost importance, because unconsidered failures will be a substantial threat for the defined safety goal.

5. ACKNOWLEDGMENT

This work was partially supported by the EU under the FP7-ICT programme, through project 288195 "Kernel-based ARchitecture for safetY-critical cONtrol" (KARYON).

References

[1] J. Shi, J. Wan, H. Yan, and H. Suo, "A Survey of Cyber-Physical Systems," in *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*, IEEE, 2011, pp. 1–6.

[2] D. Cook and S. Das, "How smart are our environments? An updated look at the state of the art," *Pervasive and Mobile Computing*, vol. 3, no. 2, pp. 53–73, 2007.

[3] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini, "A Survey of Context Data Distribution for Mobile Ubiquitous Systems," *ACM Computing Surveys*, vol. 45, no. 1, pp. 1–49, 2013.

[4] S. Zug, A. Dietrich, and J. Kaiser, "Fault diagnosis in robotic and industrial systems," in *Fault Diagnosis in Robotic and Industrial Systems*, G. Rigatos, Ed. St. Franklin, AUS: Concept Press Ltd., 2012, ch. Fault-Handling in Networked Sensor Systems.

[5] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, and M. Srivastava, "Sensor network data fault types," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 3, p. 25, 2009.

[6] The Mathworks, *Simulink - Webseite*, 2011. [Online]. Available: \url{http://www.mathworks.de/products/simulink/}.

[7] O Lisagor, J. McDermid, and D. Pumfrey, "Towards a practicable process for automated safety analysis," in *24th International System Safety Conference*, Citeseer, 2006, pp. 596–607.

[8] R. K. I. Mei-Chen Hsueh Timothy K. Tsai, "Fault injection techniques and tools," *A World of Wireless, Mobile and Multimedia Networks, International Symposium on*, vol. 30, pp. 75–82, 1997.

[9] H. M. Raul Barbosa Johan Karlsson and M. Vieira, "Fault injection," in *Resilience Assessment and Evaluation of Computing Systems*. 2012.

[10] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, and G. H. Leber, "Comparison of physical and software-implemented fault injection techniques," *Computers, IEEE Transactions on*, vol. 52, no. 9, pp. 1115–1133, 2003.

[11] M. Blanke, M. Staroswiecki, and N. E. Wu, "Concepts and methods in fault-tolerant control," in *American Control Conference, 2001. Proceedings of the 2001*, IEEE, vol. 4, 2001, pp. 2606–2620.

[12] R. Svenningsson, J. Vinter, H. Eriksson, and M. Törngren, "Towards fault injection based minimal cut sets generation," in *Informationstagung Mikroelektronik ME-10*, 2010.

[13] A. Joshi and M. P. Heimdahl, "Model-based safety analysis of simulink models using scade design verifier," in *Computer Safety, Reliability, and Security*, Springer, 2005, pp. 122–135.

[14] P. Bieber, C. Castel, and C. Seguin, "Combination of fault tree analysis and model checking for safety assessment of complex system," in *Dependable Computing EDCC-4*, Springer, 2002, pp. 19–31.

[15] M. Bozzano, A. Villafiorita, O Åkerlund, P. Bieber, C Bougnol, E Böde, M Bretschneider, A Cavallo, C Castel, M Cifaldi, *et al.*, "Esacs: an integrated methodology for design and safety analysis of complex systems," in *Proc. ESREL*, 2003, pp. 237–245.

Annex F Driving with Confidence: Local Dynamic Maps That Provide LoS for the Gulliver Test-Bed

C. Berger, O. Morales, T. Petig and E. M. Schiller, "Driving with Confidence: Local Dynamic Maps That Provide LoS for the Gulliver Test-Bed," in *Computer Safety, Reliability, and Security*, Springer, 2014, pp. 36-45.

This page is intentionally left blank.

Driving with Confidence: Local Dynamic Maps That Provide LoS for the Gulliver Test-Bed[⋆]

Christian Berger, Oscar Morales, Thomas Petig, and Elad Michael Schiller

Computer Science and Engineering,
Chalmers University of Technology, Sweden
{christian.berger,mooscar,petig,elad}@chalmers.se

Abstract. The design of automated driving systems aims at reducing the human error and increasing the fuel efficiency by letting the vehicles map their surroundings and drive autonomously. One of the system challenges on the road is that at any time the environment can stop meeting the system's operational conditions (and then resume meeting the requirements at some later point in time). Thus, as vehicles map their surroundings, they should also provide information that can help the vehicles to know whether the operational conditions are met with respect to the confidence that they have about the mapped information.

We design and implement key services of *Local Dynamic Maps* (LDMs) that are based on on-board and remote sensory information. The LDM provides the position of all nearby noticeable objects along with the LDM's confidence about these positions. The design also includes an extension that allows the vehicular system to agree on the lowest common ability to meet the operational conditions.

We evaluate the performance of a key component in our pilot implementation together with a set of test cases that validate the proposed design. Our current findings show that the presented ideas can accelerate the deployment of automated driving systems.

1 Introduction

Self-driving cars will be the next big step in vehicular technology as several important automotive original equipment manufacturers (OEMs) have recently announced [9]. However, their specific challenge besides deploying a robust and reliable technology throughout a vehicle's lifetime [5] is to bring down the technology's costs. Therefore, expensive sensors that perceive a vehicle's surroundings need to be substituted by cheaper counterparts. Cheap sensors normally have a reduced accuracy. This is addressed by sensor fusion with information provided by other vehicles and the infrastructure.

[⋆] The work of this author was partially supported by the EC, through project FP7-STREP-288195, KARYON (Kernel-based ARchitecture for safetY-critical cONtrol).

A. Bondavalli et al. (Eds.): SAFECOMP 2014 Workshops, LNCS 8696, pp. 36–45, 2014.

Research in this area however is time-consuming, error-prone, expensive, and tedious, when several cars need to be coordinated within a real-scale experiment on a real proving ground. As an intermediate for instance, preliminary experiments can be planned and conducted with miniaturized counterparts. We maintain such a fleet of scaled autonomous and cooperative vehicles using the Gulliver Testbed [15]. Different use cases with our test-bed have successfully shown [2, 3] that it is possible to bridge between purely virtual experiments as carried out in simulations and physical experiments on real-scale proving grounds [4].

Our system design has two distinct parts that each has different timing properties, following the architectural hybridization concept [8]. Given the uncertainties affecting the system operation and the confidence in the data used in control processes, we use the architectural concept of safety kernel. This concept is responsible for managing the task, in a way, that ultimately ensures the required safety goals. The vehicle limited ability to communicate prevents centralized solutions and open the door to cooperative ones. We consider sensory data that has validity attributes attached that defines that accuracy and conference in the data. The (decartelized) safety kernel uses these attributes to decide on a system service level that in turn will set the system performance level after cooperatively evaluating the service level. This version of the paper refers to the work that was done in KARYON with respect to local dynamic maps. We note that cooperation to construction of localization maps was earlier discussed in other projects, such as Hidenets.[1]

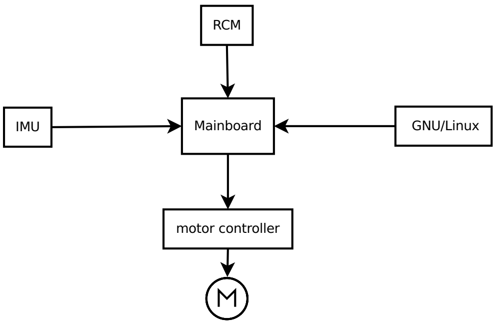


Fig. 1. Gulliver vehicle (hardware) architecture

We have designed and implemented the Gulliver test-bed [4, 15] with an emphasis on demonstrating safety aspects of cooperative systems, and system architecture to the concrete implementation of fundamental components. The software architecture within each vehicle follows the proposed architectural pattern and, in particular, uses a safety kernel for safety management. For that, the hardware and software solution presented in this paper are based on an earlier design in which we have implemented and integrated the safety kernel in Gulliver vehicles [8]. Thus, the test-bed is adequate to demonstrate the architectural concept, and to show that it is possible to manage the performance level depending on the operational conditions while ensuring that the functions always perform safely.

[1] www.hidenets.aau.dk

2 System Overview

We present the key implementation issues of the Gulliver test-bed, which we have further developed based on an earlier design [4, 15].

Hardware Architecture. The hardware architecture is sketched in Figure 1. The central component is the mainboard. Further components are the inertial measurement unit (IMU), the ranging device (RCM), a GNU/Linux system and the motor controller.

The IMU provides heading information that is derived from a gyroscope. The RCM provides position information and allows communication between the mainboards of different vehicles. The GNU/Linux system supervises the operation of the vehicle and provides a platform for cooperative algorithms. The vehicles can communicate with each other and the test-bed control client via Wifi links. The motor and the steering servos are controlled by the motor controller. Additionally, it provides odometry information.

Further vehicles from the Gulliver Testbed [2, 3] comprise components that also enable experiments for self-driving vehicular technology. These vehicles participate in the annual international competition CaroloCup[2] for miniature self-driving cars.

Localization. The localization system is based on two different sensors, a ranging device and an inertial measurement unit. The ranging device is P410 RCM produced by timedomain. It uses an ultra-wide band transceiver and measures the time of flight between two modules. Therefore, three stationary anchors are used as reference points. They have a known position and define the reference frame. The ranging devices are sharing a common wireless and, hence, we have the schedule of the transmissions. Our self-stabilizing approach is presented in [16] and [17]. It features a TDMA timeslot assignment algorithm that does not utilize an external reference.

The position is estimated from the ranging outcome, the odometry, and gyroscope data by a Kalman filter [18]. We have studied the influence of reflections and interferences on our localization system. We have experimented both outdoor and indoor settings, see figures 2 and 3, respectively. We have used these results when designing the safe distance that vehicles should keep from each other when driving in the test-bed.

Path Planning and Following. The Gulliver demonstrator uses a set of predefined paths during a demonstration. The paths are defined by the operator especially for the application that is to be demonstrated.

A waypoint is defined as (x, y, v), where $x, y \in \mathbb{Z}$ is the position of the waypoint on the plane and $v \in \mathbb{Z}$ is the proposed maximum speed used to reach this waypoint. The vehicles follow predefined paths; each is a finite ordered sequence of waypoints, where the last waypoint follows the first.

For some test cases, e.g., it is useful to define multiple waypoints. Thus, we support several paths and we allow the vehicle to switch between them.

[2] www.carolocup.de

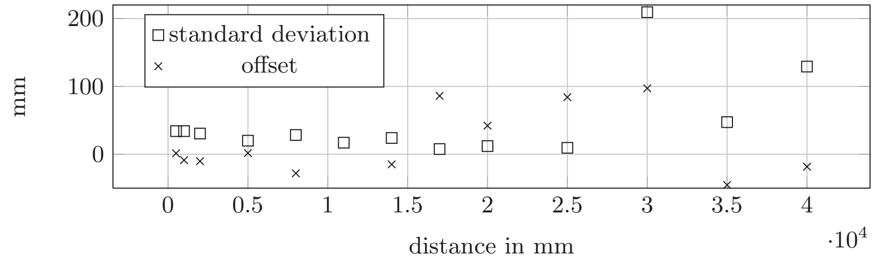


Fig. 2. Outdoor accuracy of the RCM. The offset (difference of measured mean and actual distance) and the standard deviation of 10k measurements each.

External Vision Based Localization. An external localization system can help to supervise the operation of the demonstrator. A vision based system can give, after calibration, absolute coordinates of the vehicles with respect to a given reference frame. Our system uses inexpensive standard USB cameras as external references. Each vehicle is equipped with a unique tag that can be recognized by image processing software.

We are using OpenCV, a software toolkit that was originally introduced in [7] as CVLib. It provides a programming interface for acquiring frames from the camera, as well as composable algorithms for image processing. The vehicles are equipped with unique AprilTags [14]. These tags allow the vision-based localization system to compute the vehicle id, position and orientation. The computation is done for every frame separately. The position of the camera is automatically determined by a group of four reference tags on the floor with known positions. These can be used to compute a perspective transformation matrix P. Using this matrix, a vehicle's position can be computed directly from the coordinates in the captured frame. The resulting vehicle positions are sent in User Datagram Protocol (UDP) to the test-bed control client and integrated with LDM, as well as with the Gulliver software.

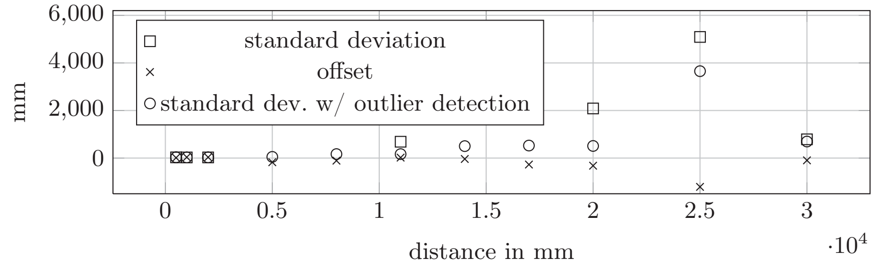


Fig. 3. Indoor accuracy of the RCM. The offset (difference of measured mean and actual distance) and the standard deviation of 10k measurements each.The standard deviation increases due to reflections.

3 Local Dynamic Map

We present our design for a Local Dynamic Map (LDM) that is inspired by `ETSI TR 102 863` and focus mainly on highly dynamic information (type 4). We follow KARYON's view on confidence, with respect to position, heading, speed, etc., and provide data validity information that includes time, offset and outlier [6]. Our pilot focuses merely on the position data from on-board sensors, as well as sensory information that can be collected from nearby vehicles.

Since remote sensory information is prone to communication interferences and delays, we use a hybrid architecture, in which the architecture is divided into one real-time part and another in which complex computations are allowed, such as vehicle-to-vehicle communication. Thus, the system can always rely on a baseline service that is provided by on-board sensors. When the opportunity occurs and the operational conditions improve, the system upgrades its performance by using remote sensory information for gaining more confidence. Note that one of the key advantages of this hybrid approach is that the system design does not require the access to communication systems that never fail (or with very high probability). In case that those communication failures bring the confidence level below the operational requirements, the system can always rely on the baseline service until better confidence is gained and the system can upgrade its service. Our design assumes the existence of a safety kernel that sets the system performance level according to the recent events [8].

On-board Local Dynamic Map. The on-board part uses merely on-board sensors that can be implemented in a real-time manner. On-board maps, for instance, are built and updated while the vehicle is driving through an unknown or previously mapped environment to realize a self-localization and mapping algorithm. Sensors that can be used for this purpose include: rotary encoders like wheel encoders, incremental encoders, hall-effect sensors, mice-based odometers; distance sensors like ultra-sonic sensors, infrared sensors, or depth sensors like laser or radar sensors; even vision sensors can be used to analyze the optical flow for instance. In combination with an IMU device, the input data from such sensors is fused and integrated over time to create and update local onboard maps. However, without regular updates from an external reference system, such onboard-only systems are affected by increasing data error because the used models, for such onboard maps, drift over time as inaccuracies in the measured data can occur for instance.

Cooperative Local Dynamic Map. This network-oriented part collects the position information from nearby vehicles, such as position, speed, and heading together with the data age. The Cooperative LDM provides timing information (TFD data) for the timing failure detector (TFD) and validity. The TFD data allows the TFD to detect the liveliness of the Cooperative LDM. Note that this does not contain information about the data age that is collected from other vehicles. The validity contains information on how certain the Cooperative LDM is about the position information stored, whereas certainty is meant over the coordinates and time.

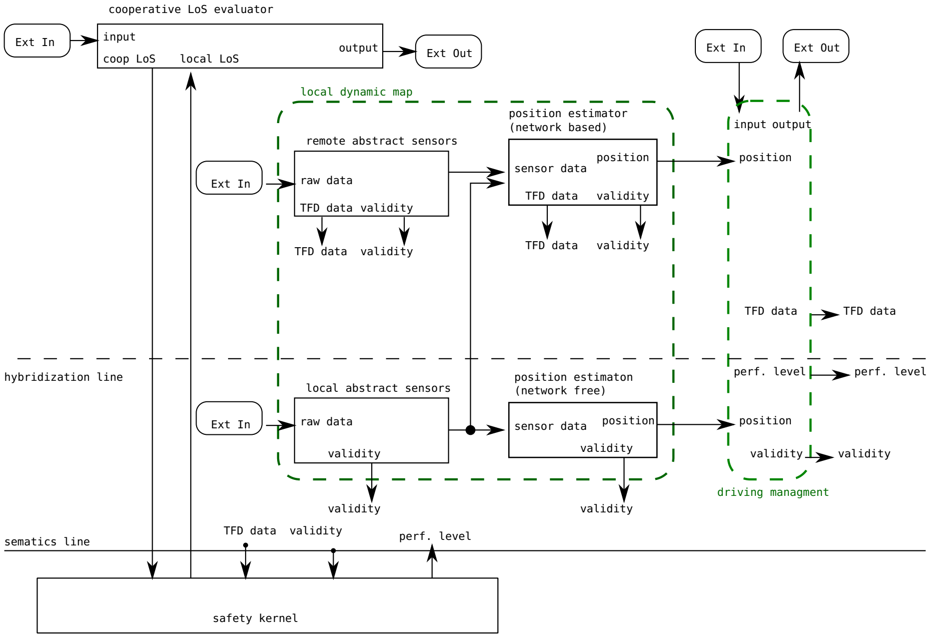


Fig. 4. Gulliver software architecture of a single vehicle

4 Cooperative Vehicular Algorithms

We selected test cases for which we can define two applications; a fully cooperative one that we associate with the highest service level, and autonomous one that we associate with the lowest service level. We explain how the vehicles' driver manager can act upon the operation service level, which the Cooperative Service Level Evaluator can provide. We present a pilot implementation for this feasibility study.

Test Cases. We considered three test-cases for performing the experiments.[3] For the completeness sake, this paper includes a brief description of these test cases. More details can be found in [1].

Adaptive cruise control and vehicular platooning. Vehicles maintain a safety distance from the vehicle ahead. We set to 3 sec the headway for the Vehicular Adaptive Cruise Control (lowest service level) and 1 second for platooning (Highest service level).

Intersection crossing. The highest service level application coordinates the intersection crossing so that the waiting time is minimized while the lowest

[3] See demonstration videos at `www.chalmers.se/hosted/gulliver-en/documents`

service level application maintains a conservative approach, in which vehicles stop before crossing and let the vehicle coming from the right to cross first.

Coordinated lane change. The highest service level application coordinates a lane change maneuver with minimum inter-vehicle distance while the lowest service level application considers a conservative approach in which the maneuver starts until a sufficiently large space is created.

The Driver Manager. The (decentralized) driver manager, as well as the cooperative evaluator of level service, does not rely on a distinctive vehicle or leader election. The design is based on (not necessarily aligned) *rounds* of 190 ms, which are locally divided into four phases:

*Observe (*80 *ms).* Each vehicle updates its local information (localization, speed, lane, etc.) from the mainboard and broadcasts it along with all the vehicle's localizations that it has received since the last round. The broadcast is transmitted twice with 40 ms between retransmissions.

*Compute (*10 *ms).* Each vehicle computes the trajectory for all the level of services that the vehicle supports in each test case using the acquired information since the last round. The time costs of all the advanced driver assistance systems is $O(n)$ with preprocessing time of $O(n \log(n))$, where n is the number of vehicles. During our three vehicle experiments, we observed a sub-millisecond trajectory computation cost but for redundancy reasons we assume 10 ms.

*Agreement (*80 *ms).* Each vehicle executes the cooperative service level evaluator to agree on the cooperative service level that all the vehicles will run in the next round. Essentially, each vehicle broadcasts its maximum local level of service as well as the maximum level of service from the vehicle that it has received since the last round. The broadcast is transmitted twice with 40 ms between retransmissions. Thus, the phase can be completed within 80 ms. Note that the vehicles operate in distinct level of service for no longer than two consecutive rounds.

*Move (*20 *ms).* Each vehicle determines the trajectory to operate according to the cooperative service level obtained for the current round. The trajectory is then sent to the mainboard. It takes around 10 ms to send the trajectory through the serial port to the mainboard and receive the acknowledge, but for redundancy reasons we assume 20 ms.

Cooperative Service Level Evaluator. This fault-tolerant distributed vehicular system must ensure its safe operation. Each vehicle implements a *cooperative service level evaluator* that on every round decides what would be the lowest common ability to meet the operational conditions for the next round. Therefore, the decision and its dissemination must be done in bounded time. Due to communication failures, the cooperative service level evaluator must be able to cope with participants or communication failures.

We consider n vehicles; each has a unique id. The vehicles create an ad-hoc network, i.e., no access points or base stations [10–13]. For the communication protocol, we consider UDP in order to avoid the retransmission overheads, and thus messages can be lost due to noise or interference.

The cooperative service level evaluator aims at allowing the vehicles to operate at the highest service level. It can do so when allow vehicles can support the highest service level, and the communication network delivers messages in a timely manner. Since wireless communications can experience periods of arbitrary packet drops, the cooperative service level evaluator has to lower the service level when the vehicles fail to exchange their service level reports in a timely manner. Our feasibility tests focused on the scalability of this component in ns3 and aimed at validating its behavior with respect to scenarios that involve several vehicles. For the simulation, we consider a wireless ad-hoc network with a standard channel IEEE 802.11b. We used the log distance propagation loss model with exponent 3 and reference loss of 60.0. We assume that the vehicles are deployed uniformly at random in a rectangle with dimension 30 × 150 meters. Vehicles move at a constant speed chosen randomly and uniformly between 0 and $20\frac{m}{s}$. We perform experiments with a variant number of vehicles between 2 and 30, and the number of transmissions between 2 and 4. We run each experiment for 1, 200 sec.

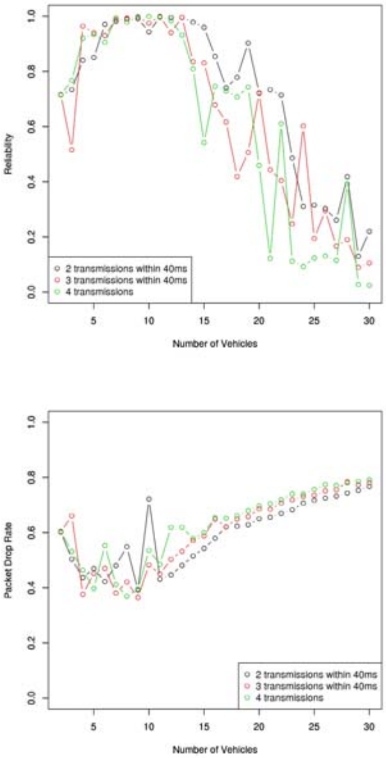


Fig. 5. Reliability of the consensus algorithm (Proportion of rounds in high service level) and packet drop rate

We estimate the reliability aspects of our implementation by considering the time that the system operates on the highest service level. We compare that time and the packet drop rate. The plot on the left of Figure 5 shows that, as the number of vehicles increases, the time that the system operates on the highest service level decreases. This is due to the increment on the packet drop rate since the medium is shared with more vehicles, as depicted on the right of Figure 5. We also validated our results via experiments that used (physical) scaled-vehicles in which the number of vehicles was between two and five. We observed that 90% of the time that the system operated on the highest service level when the number of vehicles was between two and four. This validates our computer simulations. However, there was a drop to 60% when we tested five vehicles. We believe that the reason is due to the use of network adapters from different vendors during the experiments. Further tests are needed for this case.

5 Conclusions

This paper reports on the progress of the development work. The development outlook includes further implementation of the different data validity mechanisms as well as scalable algorithms for achieving cooperative service level evaluation.

References

[1] Casimiro, A., Oscar Morales-Ponce, T.P., Schiller, E.M.: Vehicular coordination via a safety kernel in the gulliver test-bed. In: The Thirteenth International Workshop on Assurance in Distributed Systems and Networks (ADSN 2014). IEEE (2014)

[2] Berger, C.: From a Competition for Self-Driving Miniature Cars to a Standardized Experimental Platform: Concept, Models, Architecture, and Evaluation. Journal of Software Engineering for Robotics 5(1), 63–79 (2014)

[3] Berger, C., Al Mamun, M.A., Hansson, J.: COTS-Architecture with a Real-Time OS for a Self-Driving Miniature Vehicle. In: Schiller, E.M., Lönn, H. (eds.) Proceedings of the 2nd Workshop on Architecting Safety in Collaborative Mobile Systems (ASCoMS), Toulouse, France, pp. 1–12 (September 2013), http://hal.archives-ouvertes.fr/docs/00/84/81/01/PDF/00010133.pdf

[4] Berger, C., et al.: Bridging physical and digital traffic system simulations with the gulliver test-bed. In: Berbineau, M., Jonsson, M., Bonnin, J.-M., Cherkaoui, S., Aguado, M., Rico-Garcia, C., Ghannoum, H., Mehmood, R., Vinel, A. (eds.) Nets4Trains/Nets4Cars 2013. LNCS, vol. 7865, pp. 169–184. Springer, Heidelberg (2013)

[5] Berger, C., Rumpe, B.: Autonomous Driving - 5 Years after the Urban Challenge: The Anticipatory Vehicle as a Cyber-Physical System. In: Goltz, U., Magnor, M., Appelrath, H.J., Matthies, H.K., Balke, W.T., Wolf, L. (eds.) Proceedings of the INFORMATIK 2012, pp. 789–798. Braunschweig, Germany (2012)

[6] Brade, T., Zug, S., Kaiser, J.: Validity-based failure algebra for distributed sensor systems. In: SRDS, pp. 143–152. IEEE (2013)

[7] Bradski, G.R., Pisarevsky, V.: Intel's computer vision library: Applications in calibration, stereo, segmentation, tracking, gesture, face and object recognition. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition, vol. 2, p. 2796 (2000)

[8] Casimiro, A., Rufino, J., Pinto, R.C., Vial, E., Schiller, E.M., Morales-Ponce, O., Petig, T.: A kernel-based architecture for safe cooperative vehicular functions. In: 9th IEEE International Symposium on Industrial Embedded Systems, SIES 2014 (2014)

[9] Hirsch, J.: Self-driving cars inch closer to mainstream availability (October 2013), http://www.latimes.com/business/autos/la-fi-adv-hy-self-driving-cars-20131013,0,5094627.story

[10] Leone, P., Papatriantafilou, M., Schiller, E.M.: Relocation analysis of stabilizing MAC algorithms for large-scale mobile ad hoc networks. In: Dolev, S. (ed.) ALGOSENSORS 2009. LNCS, vol. 5804, pp. 203–217. Springer, Heidelberg (2009)

[11] Leone, P., Papatriantafilou, M., Schiller, E.M., Zhu, G.: Chameleon-MAC: Adaptive and self-$\star$ algorithms for media access control in mobile ad hoc networks. In: Dolev, S., Cobb, J., Fischer, M., Yung, M. (eds.) SSS 2010. LNCS, vol. 6366, pp. 468–488. Springer, Heidelberg (2010)

[12] Leone, P., Schiller, E.M.: Self-stabilizing TDMA algorithms for dynamic wireless ad-hoc networks. Int. J. Distributed Sensor Networks, 639761 (2013)

[13] Mustafa, M., Papatriantafilou, M., Schiller, E.M., Tohidi, A., Tsigas, P.: Autonomous TDMA alignment for VANETs. In: 76th IEEE Vehicular Technology Conf. (VTC-Fall 2012), pp. 1–5. IEEE (2012)

[14] Olson, E.: AprilTag: A robust and flexible visual fiducial system. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 3400–3407. IEEE (May 2011)

[15] Pahlavan, M., Papatriantafilou, M., Schiller, E.M.: Gulliver: a test-bed for developing, demonstrating and prototyping vehicular systems. In: Proceedings of the 9th ACM International Symposium on Mobility Management and Wireless Access, pp. 1–8. ACM (2011)

[16] Petig, T., Schiller, E.M., Tsigas, P.: Self-stabilizing tdma algorithms for wireless ad-hoc networks without external reference. CoRR abs/1308.6475 (2013)

[17] Petig, T., Schiller, E.M., Tsigas, P.: Self-stabilizing TDMA algorithms for wireless ad-hoc networks without external reference. In: Higashino, T., Katayama, Y., Masuzawa, T., Potop-Butucaru, M., Yamashita, M. (eds.) SSS 2013. LNCS, vol. 8255, pp. 354–356. Springer, Heidelberg (2013)

[18] Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). The MIT Press (2005)

Annex G Distributed management and representation of data and context in robotic applications

A. Dietrich, S. Zug, S. Mohammad and J. Kaiser, “Distributed management and representation of data and context in robotic applications,” in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 2014.

This page is intentionally left blank.

**2014 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS 2014)
September 14-18, 2014, Chicago, IL, USA**

Distributed Management and Representation of Data and Context in Robotic Applications*

André Dietrich[1], Sebastian Zug[1], Siba Mohammad[2], and Jörg Kaiser[1]

Abstract— **The traditional, isolated data handling in sensor-actuator systems does not fulfill the requirements of robots that need to interact with their smart environment. Consequently, we have to develop new mechanisms for adaptive data and context handling.**

We firstly investigate what types of data are present within smart environments and how they can be classified and organized. Only if the available data can be structured, it can be queried and thus put into context. This is important because the variety of data and possible interpretations is tremendous, ranging from measurement values, sensor and robot descriptions/states/commands, to environmental data, such as positions, maps, spatial relations, etc. To cope with this diversity, we developed a solution capable of storing and accessing data within a distributed environment by providing additional context information. Furthermore, we describe how this information can be assembled in a task-oriented manner. This enables robots to dynamically generate environmental abstractions by using data from different sources and also enables them to incorporate external sensor measurements.

I. INTRODUCTION

If we think of future smart environments (a good overview is given in [1]), whether in industrial manufacturing, building automation, logistic processes, or health-care scenarios, we think of various different smart entities, capable of functioning in dynamic and changing environments, interacting with each other and solving tasks in cooperation. These entities are capable of sharing their knowledge, experiences, and environmental perception. New and evolving technologies and paradigms like the "Internet of Things" (cf. [2]), "Cyber-Physical Systems" (cf. [3]), or "Pervasive/Ubiquitous Computing" (cf. [4]) adopt these ideas and intend a flexible communication between different intelligent components. However, simply transferring data is by far not enough. It is rather the basis for flexible cooperations, or from another perspective just the "tip of the iceberg" of what needs to be done. In the end, there is a tremendous difference between a value and a meaning. What does a change of a single distance measurement stand for? Thus, meaning can only be derived by transforming, interpreting, and reinterpreting data according to the current context. Whereby, a context can be defined by everything that is relevant to fulfill a certain task, like other sensor measurements, location and infrastructure, safety and security requirements, time, physical conditions, etc. (cf. [5]). This opens up two rarely asked questions:

1) *How can data be stored and accessed in conjunction with context?*
2) *How can data, gathered from various sources, and context be dynamically put together, to deduce any kind of application specific information?*

We construe a smart environment with appearing and disappearing entities, with overlapping workspaces, changing tasks, etc., as some kind of distributed mind that is continuously producing new data, information, and knowledge. Systems in such environments will have to continuously adapt to these changes, and therefore will require holistic access to this distributed mind. Thus, there is a myriad of data (measurements, commands, states) and descriptions (metadata) resident within smart and distributed environments, which is somehow related (spatial, temporal, etc.), but it is not organized, not directly accessible, and cannot be queried (other problems are discussed in [6]).

Imagine a robotic platform that enters a (manufacturing) hall for the first time to deliver some cargo to a certain location. This platform would establish a connection to the smart entities of the environment and simply request all needed information, like a map of the hall with a sufficient level of detail, including restricted areas, present robotic systems, available sensors, human positions, etc. According to the determined route, it would then just filter out irrelevant information and possibly try to get access to external sensors along its way. The flexible usage of external sensor allows to extend the local view on the environment. The robot is able to navigate more efficiently, by preventing sudden and unpredictable dangerous situations and can increase its traveling speeds.

The common benefits of such a flexible exchange and integration of information are obvious. It provides a higher coverage and higher degree of fault tolerance compared to individual perception. At the same time, systems can interact more efficiently and coordinate their behavior. It should be noted that such future cooperations will happen between previously unknown systems and therefore have to tolerate dynamic integration and segregation of heterogeneous entities. Unlike today where cooperation or even the integration of external sensor measurements is only possible if, and only if, it was previously intended by the programmer/system-designer.

*This work is (partly) funded by the German Ministry of Education and research within the project ViERforES-II (grant no. 01IM10002B) and by the EU FP7-ICT program under the contract number 288195 "Kernel-based ARchitecture for safetY-critical cONtrol" (KARYON).

[1] A. Dietrich, S. Zug and J. Kaiser are with the Department of Distributed Systems, at the Otto-von-Guericke-Universität Magdeburg in Germany {dietrich,zug,kaiser}@ivs.cs.uni-magdeburg.de

[2] S. Mohammad is with the Database and Information Systems Group at the Otto-von-Guericke-Universität Magdeburg in Germany siba.mohammad@iti.cs.uni-magdeburg.de

978-1-4799-6933-3/14/$31.00 ©2014 IEEE

Structure and Overview

To tackle the above mentioned problems more efficiently, we subdivided our approach into three conceptual parts. Starting bottom up, we first identify what data is available in such environments and how it can be classified. Based on this preliminary investigation, we present a solution to *question 1*, which allows us to organize this data (with its spatial and temporal contexts) so that it can be dynamically accessed and queried. The third part is based mainly on our previous research efforts and describes how this data, coming from different sources, can be combined freely and how to deduce any kind of application specific information (*question 2*). Additionally, we demonstrate the applicability of our approach by replaying the introductory example on the floors of our faculty building. To illustrate important aspects of this paper in a more convenient way, we uploaded additional video material to our YouTube-channel at `http://www.youtube.com/ivsmagdeburg` and denote its appearance in some figures with "Fig.$^{\text{Ani.}}$".

Because the presented concept touches different research ideas, we present a comparison to the related work at the end of this paper, followed by the conclusion.

II. Different Flavours of Data

There is a tremendous difference between the often erroneously mixed terms knowledge, information, and data, which is caricatured in Fig. 1. As described in [7] (in comparison to different research areas), data is mostly assumed as simple, discrete, and isolated facts without any explicit interpretation (e. g., the positions and trajectories of robots and humans). Combining data within a structure or putting it into a certain context generates information (e. g., combining the trajectories and current positions within a map). Giving information a meaning by interpreting it, produces knowledge (e. g., the prediction of a collision, due to intersecting trajectories). While intelligence comes into play by choosing alternatives or by deciding on strategies based on knowledge (e. g., preventing the collision by choosing between a full-stop, a change of speed, by recalculating the trajectory, etc.).

In the context of smart environments, we can subdivide data into the following four general categories (cf. Fig. 1):

1) Metadata: it can be translated as "data about data" and refers to the static descriptions of data-formats (e. g., TEDS[3], ROS-msg[4]), entities (e. g., MOSAIC[5], SensorML[6]), or complete systems (e. g., AutomationML[7]) and interfaces. It allows to access external systems, describes their configurations, and enables the decoding of raw data. Literally, metadata is used to describe every sensory system of a robot, the robot itself, and external sensors in detail as well as how to access these systems and to decode their message streams.
2) Raw Data: it is the amount of directly measurable physical quantities that change over time, such as distance (laser scans), temperature, light (camera frames). This group furthermore comprises status messages gathered sensors or actuators.
3) Virtual Data: it can be described as indirect measurements that are derived from raw data by applying physical or mathematical laws (cf. [9]). This type is required where physical modalities are not directly measurable, such as the temperature within a combustion engine, or simply to reduce the amount of raw data and to deduce more expressive values, like a maximum temperature or the average income.
4) Abstract Data: it is higher-level and can be generated by abstracting from raw and virtual data. A wall for example or a map, which are abstracted from multiple laser scans, a detected human within an individual camera frame or deducing the human's trajectory by analyzing multiple frames.

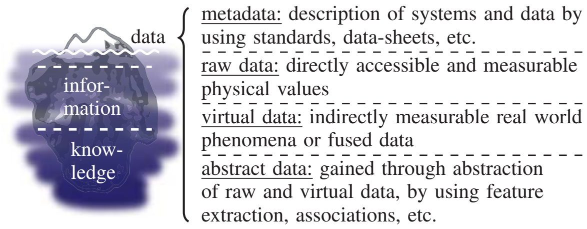

data
infor-
mation
know-
ledge
metadata: description of systems and data by using standards, data-sheets, etc.
raw data: directly accessible and measurable physical values
virtual data: indirectly measurable real world phenomena or fused data
abstract data: gained through abstraction of raw and virtual data, by using feature extraction, associations, etc.

Fig. 1: Differences of data as well as information and knowledge in smart environments.

Furthermore, raw and virtual data are useful in two ways: They have a real-time value as well as historical value. While access to real-time data is necessary for most systems that directly manipulate or interact with their environment, or bodiless applications for controlling or surveillance; a wide range of applications also require further access to historical data. Examples are the mining of sensor logs to detect unusual patterns, analysis of historical trends, post-mortem analysis of particular events, or simply for later data abstraction (mapping). Thus, the archival storage of past sensor data is becoming more important.

We therefore tried to create a minimalist solution that covers all the above mentioned types of data as well as their specific real-time/historical value and spatial and temporal context. This requires the combination of two diverse areas: Distributed robotic applications (as a part of smart environments) with distributed or nowadays cloud-based database systems. This conceptual approach is presented within the following section.

[3] "Transducer Electronic Data Sheet", which is part of the smart transducer interface standards set IEEE 1451 [8].

[4] A messages description language used to define various message formats for the ROS publish/subscribe system: `www.ros.org/wiki/msg`

[5] The "fraMework for fault-tOlerant Sensor dAta processIng in dynamiC environments" provides several XML descriptions for accessing sensors as well as actuators: `www.code.google.com/p/mosaicframework`

[6] "Sensor Model Language" offers models and encodings to describe sensors and measurement processes: `www.ogcnetwork.net/SensorML`

[7] Automation Markup Language is a description format for storing and exchanging plant engineering information. `www.automationml.org`

III. DATA ORGANIZATION AND STORAGE

The key idea is to use databases as a kind of holistic storage, where every entity hosts a local database instance to memorize any kind of data that may be usable to carry out a certain task. The usage of distributed databases should therefore allow the entity to seamlessly access and query data from all other connected entities. We decided to apply Cassandra as our prior database system in combination with ROS (Robot Operating System [10]) and its communication infrastructure and message description formats.

Within this section, we describe how data is stored and organized by keeping spatial and temporal relations. A simplified diagram revealing the basic database structure and the organization of data is depicted in Fig. 2. But firstly, we discuss some benefits of Cassandra as well as the reasons for choosing this particular database system.

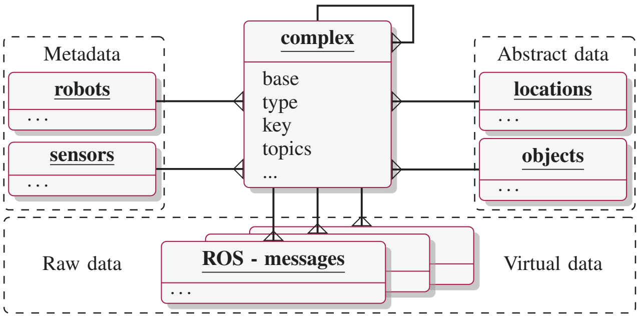


Fig. 2: Organization of data with column-family complex as the global link.

A. Why Cassandra?

Although Cassandra was initially developed as a NoSQL (Not only SQL (Structured Query Language)) database for Facebook [11] and received only little attention in the robotics community (cf. [12]), its concept as well as some of its features make it an ideal storage system for distributed applications (especially for smart environments).

Cassandra provides a distributed key-value store. Keys map to rows, which can contain a multitude of columns (values), while rows by themselves are stored in column-families (tables). See also Fig. 2 to get a better impression about the data structure in Cassandra. Further columns can be added dynamically or removed at any time, so that different rows can store different types and different amounts of values. That means that the structure of the database can change over time and adapt to varying requirements, unlike a classical relational database system. As revealed in [13] with a performance evaluation, it also seems to provide an ideal storage for sensor data. In [14] we describe a generic ROS-Cassandra interface and present an evaluation that compares our solution with the ROS MongoDB[8] implementation and the standard ROS data container rosbag[9]. This comparison reveals that Cassandra memory consumption is very close to rosbag, by keeping the ability for fast and complex querying by using CQL (Cassandra Query Language). Additionally, it showed the best overall performance.

Cassandra instances can be grouped into clusters and keyspaces, allowing an entity to host and update its own database instance. By querying local data, it also queries data from other connected entities. Different strategies for replication between instances provide a high availability of data with no single point of failure. An entity can thus leave the cluster while its data remains on other nodes. Every value is marked with a timestamp, which allows to define TTLs (Time to Live), so that data can be forgotten after a certain period of time. This usage of timestamps enables eventual consistency (see also [15]), a weaker consistency level than strict or immediate consistency, which are commonly used in relational databases, but more appropriate for distributed systems. That means that due to the distribution of data and the availability of nodes, it is guaranteed to receive a valid value but maybe not the most recent. Tunable consistency level enables application dependent refinements.

B. Metadata

There are currently two types of system described with metadata: Sensors and robots. Whereas the description of sensor and actuator messages is left out to ROS and its message description language. Metadata for robots and sensors are stored within two different column-families.

1) Sensors: We used the OpenRAVE[10] XML-description format for sensors, capable of defining various kinds of sensors (such as laser scanners, odometry, etc.). This description format was combined with the MOSAIC sensor description capabilities, which allows the definition of more realistic sensors, by including various fault models (see also [16]).

2) Robots: To characterize different robots in terms of a kinematic and dynamic description, a visual representation, and a collision model, we apply common formats such as URDF (Unified Robot Description Format [17]) and COLLADA (COLLAborative Design Activity [18]).

C. Raw & Virtual Data

Our generic cassandra_ros[11] interface, introduced in [14], allows to store, query, and request historic ROS-messages (e. g., laser scans, camera streams, etc.) of any kind and source. Data can be stored either in a binary format (for fast access), in a ROS similar manner (slower due to conversation of messages, but it allows querying and analyzing data more conveniently with Cassandra's CQL-capabilities), or in other formats, like yaml or string (having their specific advantages and disadvantages). In contrast to other data, these types of data are stored in multiple column-families, one topic per column-family. This is necessary to be able to cope with the large amount of produced data, its diversity, and to reduce replication efforts. As mentioned in Sec. III-A, every stored message is automatically labeled with a timestamp by Cassadra, which allows it to keep temporal relations.

[8] wiki.ros.org/warehousewg, [9] wiki.ros.org/rosbag

[10] OPEN Robotics Automation Virtual Environment: openrave.org

[11] Project website: www.ros.org/wiki/cassandra_ros

D. Abstract Data

Actually, we identified two basic types of abstract data that should be sufficient to allow smart entities to sustain even in complex 3D environments. These are locations and objects. Therefore, both of these data types are stored within their own column-family (cf. Fig. 2).

In our case, a location always represents a certain area, which is in general static and does not change over time. It can be represented by any kind of 2D map or any kind of 3D structure. Whereas objects can define any “thing” in the environment and come up with higher dynamics than locations, ranging from rarely moved objects such as tables or wardrobes to objects with frequently changing positions (e. g., mugs, chairs, etc.). As depicted in Lis. 1, the same object or location can be represented in different formats and with different levels of precision. A mug for example can be defined as a detailed 3D model, such as vrml (Virtual Reality Modeling Language), stl (Surface Tessellation Language), etc., or as point cloud data (pcd). The value for precision is currently a subjective chosen value. An application shall later be enabled to decide upon the required amount of precision and the appropriate format, simply by parsing column names.

Listing 1: Extract of column-family objects, with row-keys (bold), columns (blue), and values.

```
objects : {  ...
  1sx34s : {
    comment : "mug ... ",
    stl_85 : binay-data,
    pcd_85 : ascii-data },
  6yfr48 : {
    wrl_90 : xml-data,
    comment : "standard phd student table",
    ... },
```

E. Complex

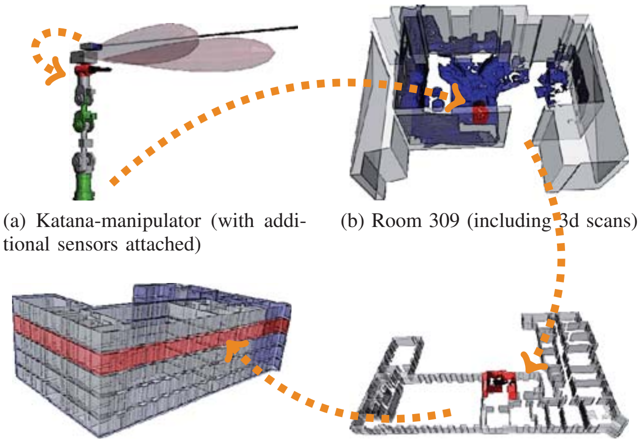
So far we have only described how data is stored, but not how this data is interconnected. To put all entities into a global context, we use a further data structure, which we call “complex”. This column-family combines all previous data (cf. Fig. 2) and organizes it within a hierarchical structure.

As listed in the code example below, a complex entry always points to a certain entity of a certain type (i. e., robot, location, sensor, object) with a certain position, identifier, etc., whereas metadata or objects are used to describe a certain class of entities. For example, there can be two mugs placed within an environment that are derived from the same (abstract-) object. A complex entry can further be used as a bodiless placeholder (to simplify some structures) by pointing onto another complex entry. Next to a position (with its specific uncertainty) and orientation relative to a base, complex entities are further labeled with communication specific information. Since we are using ROS, we store additional information about masters, topics, and services. Thus, by assigning communication details to complex entities, it is also possible to access to any kind of raw&virtual real-time data (the correct interpretation is left out to the next section) as well as to historic data, which is assigned to this topic (cf. Sec. III-C).

Listing 2: Showing the complex entry katana_62x of type robot, whose description is stored in row katana_2012 in column-family robots, whereby position and orientation are defined relative to the complex base entry room_309.

```
complex : { ...
  katana_62x : {
    key : "katana_2012",
    type : "robots",
    base : "room_309",
    position : [2.3, 3.2, 0.0],
    quaternion : [0.92, 0.0, 0.0, 32.2],
    covariance : [0.04, 0.2, 0.01, 0.14, 0...],
    ros-master : "http://moritz:11311",
    topics : ... },
  room_309 : {
    key : "room_309",
    type : "locations",
    base : "floor_4", ... },
```

The order of entities, in our case, is defined by the their positions, which are commonly defined as relative to a basis, like a robot’s position relative to a room, sensor positions and orientations mounted to a robot, or a room’s position within a floor of a building, etc. We do the same, by assigning a base to every complex entry. All positions are then defined relative to that base. Knowing the id of a single entity (bootstrapping), like the katana’s id in Fig. 3a, allows to query in two directions: Downwards, by identifying entities whose base the katana is, and upwards, by querying for the katana’s base and for entities that are related to the same base (cf. Fig. 3b).


(a) Katana-manipulator (with additional sensors attached)

(b) Room 309 (including 3d scans)

(c) Building (global base)

(d) 4th floor (assembly of multiple rooms)

Fig.[Ani. 12] 3: Hierarchy of entities, starting from the sensors whose positions are relative to the robot, while the robot is located within room 309, which is part of the fourth floor in building 29.

To sum up, it can be said that the organization of data, as we propose it, constructs a large and distributed scene graph, where every entity updates its own local Cassandra instance with positions, identified objects, sensor measurements, etc. But it is exactly what we required to solve *question 1*.

[12] Animation: www.youtube.com/watch?v=kvoC5yxdzsw

IV. DATA INTERPRETATION

Within this section, we present a solution to *question 2*, where we try to make sense of all of these different types of data and relations by transferring them into a useful representation, which we call an "environment model".

A. The Concept of an Environment Model

Environment models can be considered as an adoption of "mental models" (cf. [19]), which are widely used in cognitive science to (partially) explain how humans perceive, reason, assess, learn, and make decisions for a variety of domains. Additionally, these models are essential for integrating new information correctly. In the simplest way, they can be interpreted as "mental simulations" of the real situations or problems, with reduced complexity according to reality. Quantitative relations are mapped onto qualitative, and thus allow to store and handle elements of the world within the working memory (cf. [20]).

The term environment model is used, because we only consider information and data that might be relevant for a system (robot), to sustain in a spatial environment. While mental models are used in a much broader sense, including also sociocultural standards or knowledge and reasoning about complex action sequences. We firstly described the idea of using environment models for environmental perception and modeling in [21] and demonstrated its applicability in [22]. As depicted in Fig. 4, all data is put into a "co-simulation" of the surrounding. We currently apply OpenRAVE[10] as our prior simulation environment. It is already integrated into ROS and open source, which allows to develop own extensions[13].

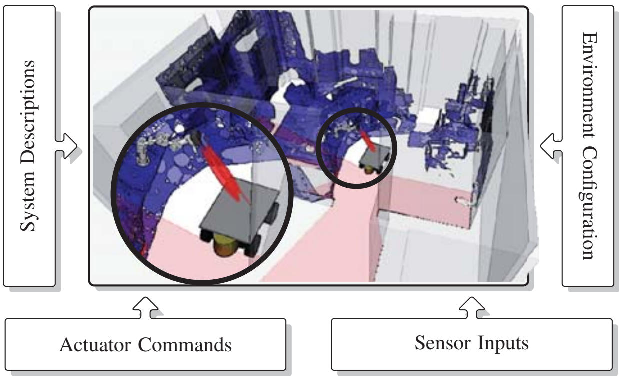


Fig. 4: Schematic structure of an environment model as an assembly of available and selected data, such as robots and sensors (metadata), real-time values including sensor measurements (red), current robot positions and configurations (raw data), as well as (abstract data) geometries of the room and depth scans (blue).

The bases for building such an environment co-simulation are the data and the relations of the last recent environmental configuration obtained from the distributed database. However, by associating the reconstructed scene and its entities with real-time values through subscribing for topics, it is possible to repeat every action and measurement within the virtual world. It allows to integrate external sensors and actuators and to interpret their in- and outputs as well as their effect on the configuration of the environment. It can be further used to validate sensor measurements by comparing their values with measurements obtained from their virtual counterparts.

All information required for applications can then be taken directly from the model, such as relative positions and distances, available sensors/robots and their location or monitoring/operational area, etc., which can be considered as abstractions of an environment model.

B. The Concept of Abstracting Views

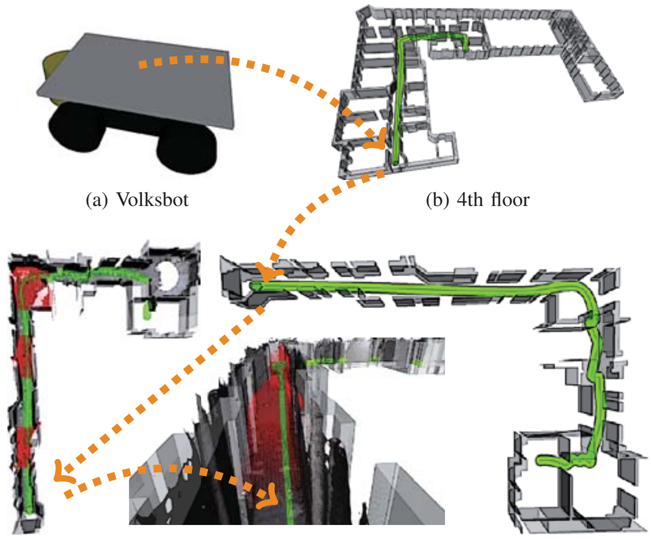
A view is a well-defined and application specific abstraction of the external environment, while the environment model is the most general representation and serves multiple applications. A view can be anything such as complex 3D models used for grasp planning or to switch perspectives (to be able to follow human orders [23]), 3D or 2D maps used for navigation and localization, or simple state vectors including distances, velocities, or even sets of objects and sensors that fulfill certain properties. Deducing such application specific information from the environment model is a quite challenging task, due to the multitude of requirements and opportunities. In [24] we discussed this problem and proposed a solution, which interprets an environment model as an implicit and dynamically changing knowledge base that can be queried in the same way as a database. We therefore had developed a new scripting semantics/language, based on simple SELECT-statements, which is mainly inspired by SQL. It even allows to define complex situations (combining time and space), which occur if the SELECT query returns a non-empty result. This query language is called SELECTSCRIPT and it exists a prototype implementation for OpenRAVE, to get a first impression on the language and its concepts see the current project web-site: `http://pythonhosted.org/SelectScript_OpenRAVE`

C. Proof of Concept

Because the benefits of our approach are hard to measure and a sequence of queries would be meaningless, we present the results of those queries and describe the conceptual way of querying (for filtering and abstracting the generated environment model we applied our query language SELECTSCRIPT). Let us go back to the delivery scenario, with which we introduced this paper. We, therefore, modified the scenario according to our local surroundings, whereby the robot does not have to deliver cargo but simply to enter `room_309`. So we start with only a model of the robot, as it is depicted in Fig. 5a, knowing its relative position to the 4th floor. By querying our system for all `locations` whose base is `floor_4`, it is possible to reconstruct a simple model of entire floor, with the correctly placed robot. This model can be used to identify the position of the target room and its relative position to the robot. Additionally, this

[13] Our plugins: `code.google.com/p/eos-openrave-plugins`

model is used for an initial trajectory planning (see Fig. 5b), which allows to it remove all `locations` that were not touched by the calculated path afterwards (see Fig. 5d). The remaining `locations` are used as bases to query for all related sensors, robots, objects, abstracted data and historic raw measurements (like a point cloud, retrieved from an earlier Kinect scan). All of these entities are then placed within the correct spatial context into the model, compare with Fig. 5c.



(a) Volksbot (b) 4th floor

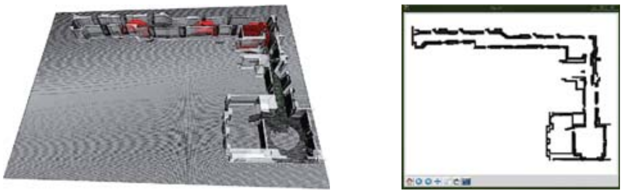
(c) Complex environment (d) Trajectory filtered locations

Fig.[Ani. 14] 5: Query sequence: (a) starting from the robot model and its relative position; (b) adding location information about the 4th floor, which is used for trajectory planning; (d) the resulting trajectory is used as a filter to segregate information about not required locations; (c) integration of entities, such as external sensors (red) and Kinect scans.

The resulting model can be used afterwards in multiple ways. On the one side it allows to identify and retrieve all relevant information about the current operational area, about included objects, robots, sensors, and their current configurations. But on the other side, it can be further abstracted to generate application specific views. The entire environment model is far too complex and exaggerated for the purpose of navigation. Thus, a simple occupancy grid map might be the better choice. As depicted in the screen shots in Fig. 6, something like maps can be simply generated by applying different filters.

V. Comparison with Related Work

Sharing data between entities is not a new idea, in contrast to its organization and structuring for smart environments. The outcome is that data gained from different sources can be reused, combined and transformed into more meaningful representations. Our notion of a meaningful representation of data is based on environment models (cf. Sec. IV-A). These models are used for two purposes, they are required for including external measurements, which requires knowledge about the type of sensor, relative positions, etc., but also to integrate newly made observations. In concrete terms, an autonomous robot should be able to look around the corner, by using external cameras to detect humans and other obstacles, but it should also be able to share this experience.



(a) Environment model with filter (b) Occupancy grid map

Fig.[Ani. 14] 6: Abstraction of maps from complex models by applying filter-functions.

As the following overview reveals, there is some related research that implicitly presents solutions our initial *questions 2* and *1*. But it also reveals that the growing complexity of environment models and the ability for distribution are also an outcome of previous developments and future requirements.

A. *Simple One-Purpose Models*

The differences in tasks and environments in robotics applications lead to a diversity of world models. An early overview on the state-of-the-art given in [25] reveals that most of these models were designed for specific (robotic) applications only. Examples are “Constructive Solid Geometry” [26], a geometric world model based on primitive geometrical objects, occupancy grid maps [27], octrees [27], etc. All of these examples are directly linked to the available sensor information and its respective data formats. The authors of [28] present a more specialized 3D occupancy grid map, which can be customized in terms of update rates and accuracy, to serve different needs. Other abstractions with more than just spatial semantics are not considered.

B. *Complex Multi-Purpose Models*

Nowadays there is a shift to more general and complex models, resulting from the fact that robots and their environments themselves have become more complex. New developments should serve various purposes, like decision-making, trajectory planning, obstacle avoidance, etc.

A theoretical concept of such a multi-purpose environment model was discussed in [29], its architecture adopts the psychological principals for artificial perception and consciousness. The concept differentiates between two levels of consciousness and two models. The first level of consciousness simply describes basic and reflexive (hard-wired) behavior that reacts directly to sensor inputs. Whereas, the second level uses two types of abstractions, a self and a world model, generated from system-knowledge and sensory inputs. Higher level planning and predicting is cut off from all sensory inputs and uses only the models. Unfortunately, there is no further explanation of how such models look like or what aspects of the environment they describe.

[14] Animation: `www.youtube.com/watch?v=Xt403wPCYD8`

In contrast to this, Hsiao and Roy presented an environment model in [30] and [23] that describes intersecting workspaces of robots and humans as a 3D physics simulation. In fact, this approach was a source of inspiration for our concept of co-simulating the environment. Their approach is formally based on ODE[15], a high performance library for simulating rigid body dynamics. It allows to describe an environment and the objects within in terms of shapes, masses, velocity, forces, and colors. These properties were used for enhanced predictions and it was demonstrated that the cooperation between humans and robots requires such a complex environmental representation. It enables a robot to change its own viewpoint and to interpret human commands, such as "Give me the blue screwdriver on my left!", correctly. But this environment model is built upon a fixed and centralized simulation that does not allow the dynamic integration of entities, and it does not consider specialized entities (external robots and sensors).

The idea that an environment model can be constructed from multiple sources is presented in [31]. It describes a vehicle's road environment conceptually as a (very specialized) object-oriented model. It uses a-priori information and information obtained from on-board sensors or from through car2car communication. That means that the awareness of another car in front can be obtained either from the local sensor system or from the communicated position of the front car itself. Types of entities are restricted to vehicles, pedestrians, and traffic signs, represented as 2D points on a lane. This type of modeling is ideal for situation assessment, because all required information is already translated into a simplified structure with some semantics. However, the environment of an autonomous robot is far more complex.

A concept of a complex environment model for autonomous systems, which reflects our notion, was presented in [32], [33]. It separates between sensor data, a world model, and knowledge. Sensor data is analyzed with the help of already existing knowledge, and the resulting information is passed to the world model. Knowledge is defined in terms of specific methods and algorithms for analyzing sensor data. The world model consists of objects (labeled with attributes), representing entities of interest. These objects are interconnected in a scene via relations. It remains unclear how data, knowledge, and the world model are stored or how scenes, including all details about the environment, are represented. But it reveals that there is a need for a symbolic abstractions/level to describe situations, similar to the examples within the next subsection.

C. Logic-Based Models

Models based on logic can are another type of environment modeling. Whereby, such systems already work on abstracted sensor data in terms of predicates and rules, which can be queried easily according to various aspects. This approach is mainly used to determine complex action sequences as well as to describe complex situations. In most cases it is based on the situation calculus [34]. An example is "alGOl in LOGic" better known as GOLOG [35] with its specialized dialects.

KnowRob [36] is another example of a knowledge processing system, based on Prolog and OWL (Web Ontology Language). It offers tools for the automated acquisition of concepts through observation and experience, which can be used for learning and reasoning. Perception modules therefore create 3D environment maps, track human motions, segment objects, and record robot activities, which means that all data is already abstracted by a fixed set of algorithms into a fixed set of concepts. Thus, sensor data is lost after this procedure and cannot be used for later analysis. Abstracted information is put into a knowledge base and can be easily queried afterwards. Next to its connection to RoboEarth (see next subsection) it also shows that relations can be directly extracted from a spatial model. For example frequently changing relations, such as "on", "in" or "below", are completely determined by the positions of these objects. As the authors argue themselves, storing an object's positions and all possible relations within a knowledge base would cause too much overhead, calculating the relations on demand is more elegant. We simply extend this consideration, if we claim that also all other information/relations can be extracted from a complex environment model, as discussed in Sec. IV.

D. Distribution

A distributed scene graph with uncertainty support was presented in [37]. It is intended to be used as a shared world model for robotic applications. This approach is specialized onto geometrical representations and spatial relations with support for semantic annotations. It does not allow to share further sensory data between entities, which could be used afterwards to identify or generate additional elements, that could be integrated into the scene graph.

RoboEarth [38] can be considered as a top-down approach, closely related to the questions of what data is required to support KnowRob. Whereby, we started bottom-up by querying what types of data are available and how they can be organized. RoboEarth is used to store and access the required (abstract) concepts, such as robots, objects (i. e., CAD models, point clouds, images), environments (i. e., maps and information on coordinate systems), and action recipes. This data is separated in two distinct database systems, one based on Apache Hadoop[16], for storing data hierarchically and a graph database for storing complex semantic relations between data. Thus, its main purpose lies in reasoning and in sharing knowledge between robots, but it does not allow to share sensor data and functionality.

The PEIS-Ecology (Physically Embedded Intelligent System) supports the idea of distributed robots (cf. [39]). Actors, sensors, and everyday objects are able to share a predefined set of functionalities dynamically, to serve tasks, they cannot fulfill on their own. All systems and functionalities are therefore described with metadata, which allows to determine their adequate combination or execution of functions.

[15] Open Dynamics Engine: `www.ode.org`

[16] Project website: `hadoop.apache.org`

A combination of heterogeneous and distributed robots was also presented in [40] for outdoor scenarios. It combined sensor information from different robots and used it for localization, mapping, and path planning. A similar approach was made by DAvinCi [41]. In contrast to the previous attempt, it was built on a cloud-based service[16]. A heterogeneous set of robots uploads its data that is afterwards combined for map building and segmentation. The main problem in both approaches is, that they imply that all heterogeneous robots are able to share a global 2D map, which requires that all robots are equipped with sensor systems at the same height, used for localization. Otherwise, maps[17], generated at different heights for the same location, differ too much. It is thus more appropriate to be able to share sensor data from the same location and to generate maps afterwards (cf. [42]).

VI. Conclusion

A holistic access to data in smart environments requires some kind of organization and the ability for interpretation. These facts are mostly neglected in related research efforts. Therefore, we started out by examining, what are relevant types of data in smart environments and how this data can be organized. Our solution is formally based on Cassandra, allowing every entity to store its data for its own purpose, but also to share it with interested entities. This allows it to extract any information afterwards, whereby we propose the usage of environment models and views.

References

[1] D. Cook and S. Das, "How smart are our environments? An updated look at the state of the art," *Pervasive and Mobile Computing*, vol. 3, pp. 53–73, 2007.

[2] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, pp. 2787–2805, 2010.

[3] J. Shi, J. Wan, H. Yan *et al.*, "A Survey of Cyber-Physical Systems," in *Proc. of the Intl. Conf. on Wireless Communications and Signal Processing*, 2011.

[4] P. Bellavista, A. Corradi, M. Fanelli *et al.*, "A Survey of Context Data Distribution for Mobile Ubiquitous Systems," *ACM Computing Surveys*, vol. 45, pp. 1–49, 2013.

[5] A. Dey, D. Salber, and G. Abowd, "A Context-based Infrastructure for Smart Environments," GeorgiaTech, Tech. Rep., 1999.

[6] S. Remy and M. Blake, "Distributed Service-Oriented Robotics," *IEEE Internet Computing*, vol. 15, pp. 70–74, 2011.

[7] I. Tuomi, "Data is more than knowledge: Implications of the reversed knowledge hierarchy for knowledge management and organizational memory," in *Proc. of the 32nd Annual Hawaii Intl. Conf. on System Sciences (HICSS-32)*, 1999.

[8] *IEEE Standard for a Smart Transducer Interface for Sensors and Actuators (IEEE 1451.0)*, IEEE Std., 2007.

[9] S. Kabadayi, A. Pridgen, and C. Julien, "Virtual Sensors: Abstracting Data from Physical Sensors," in *Proc. of the Intl. Symposium on on World of Wireless, Mobile and Multimedia Networks*, 2006.

[10] M. Quigley, K. Conley *et al.*, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[11] A. Lakshman and P. Malik, "Cassandra - A decentralized structured storage system," *Operating systems review*, vol. 44, pp. 35–40, 2010.

[12] S. Vijaykumar and S. Saravanakumar, "Future Robotics Database Management System along with Cloud TPS," *Intl. Journal on Cloud Computing: Services&Architecture (IJCCSA)*, pp. 431–438, 2011.

[13] J. van der Veen, B. van der Waaij, and R. Meijer, "Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual," in *Proc. of the 5th Intl. Conf. on Cloud Computing (CLOUD)*, 2012, pp. 431–438.

[14] A. Dietrich, S. Mohammad, S. Zug, and J. Kaiser, "ROS Meets Cassandra: Data Management in Smart Environments with NoSQL," in *Proc. of the 11th Intl. Baltic Conference (Baltic DB&IS)*, 2014.

[15] W. Vogels, "Eventually consistent," *ACM Communications*, 2009.

[16] S. Zug, M. Schulze, A. Dietrich, and J. Kaiser, "Programming abstractions and middleware for building control systems as networks of smart sensors and actuators," in *Proc. of Emerging Technologies in Factory Automation (ETFA '10)*, Bilbao, Spain, 2010.

[17] W. Meeussen, J. Hsu, and R. Diankov. (2012, 4) URDF - Unified Robot Description Format. [Online]. Available: www.ros.org/wiki/urdf

[18] R. Diankov, R. Ueda, and K. Okada, "COLLADA: An Open Standard for Robot File Formats," 2011. [Online]. Available: www.jsk.t.u-tokyo.ac.jp/rsj2011/_downloads/2Q1-5.pdf

[19] D. Meadows, W. Behrens, D. Meadows *et al.*, *Dynamics of Growth in a Finite World*. Camebridge, MA: Wright-Allen Press, 1974.

[20] P. Johnson-Laird, *Mental models: Towards a cognitive science of language, inference, and consciousness*. Harvard Univ. Press, 1986.

[21] A. Dietrich, S. Zug, and J. Kaiser, "Towards Artificial Perception," in *SAFECOMP 2012 Workshops*. Springer-Verlag Berlin, 2012.

[22] ——, "Geometric Environment Modeling System," in *Conf. on Manufacturing Modelling, Management and Control (IFAC)*, 2013.

[23] D. Roy, K. Hsiao, and N. Mavridis, "Mental Imagery for a Conversational Robot," *IEEE Transactions on Systems, Manufactoring and Cybernetics*, vol. 34, pp. 1374–1383, 2004.

[24] A. Dietrich, J. Kaiser, S. Zug *et al.*, "Application Driven Environment Representation," in *The 7th Intl. Conf. on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM)*, 2013.

[25] E. Angelopoulou, T. Hong, and A. Wu, "World model representations for mobile robots," in *Proc. of the Intelligent Vehicles Symp. (IV)*, 1992.

[26] A. Requicha and R. Tilove, "Mathematical foundations of constructive solid geometry: General topology of closed regular sets," Production Automation Project, Univ. Rochester, Tech. Rep., 1978.

[27] S. Thrun, "Learning Occupancy Grid Maps with Forward Sensor Models," *Autonomous robots*, vol. 15, pp. 111–127, 2003.

[28] R. Harle and A. Hopper, "Dynamic world models from ray-tracing," in *Proc. of the 2. Annual Conf. on Pervasive Computing and Communications (PerCom)*, 2004.

[29] H. Caulfield and J. Johnsonb, "Artificial Perception and Consciousness," in *Proc. of the 6th Intl. Conf. on Education and Training in Optics and Photonics*, 2000.

[30] K. Hsiao, N. Mavridis, and D. Roy, "Coupling perception and simulation: Steps towards conversational robotics," in *Proc. to the Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2003.

[31] A. Furda and L. Vlacic, "An object-oriented design of a world model for autonomous city vehicles," in *Proc. of the Intelligent Vehicles Symposium (IV)*, 2010.

[32] A. Belkin, A. Kuwertz, Y. Fischer *et al.*, "World Modeling for Autonomous Systems," *Innovative Information Systems Modelling Techniques*, vol. 1, pp. 135–158, 2012.

[33] A. Kuwertz, "Towards Adaptive Open-World Modeling," Vision and Fusion Laboratory, Institute for Anthropomatics, Karlsruhe Institute of Technology (KIT), Tech. Rep., 2012.

[34] J. McCarthy, "Situations, Actions, and Causal Laws," Stanford University Artificial Intelligence Project, Tech. Rep., 1963.

[35] H. Levesque, R. Reiter, Y. Lesperance *et al.*, "GOLOG: A logic programming language for dynamic domains," *The Journal of Logic Programming*, vol. 19, pp. 59–83, 1994.

[36] M. Tenorth and M. Beetz, "KnowRob – Knowledge Processing for Autonomous Personal Robots," in *Proc. of the Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[37] S. Blumenthal, H. Bruyninckx, W. Nowak *et al.*, "A Scene Graph Based Shared 3D World Model for Robotic Applications," in *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*. IEEE, 2013.

[38] M. Waibel, M. Beetz, J. Civera *et al.*, "RoboEarth," *IEEE Robotics & Automation Magazine*, vol. 18, pp. 69–82, 2011.

[39] A. Saffiotti, M. Broxvall, B. Seo *et al.*, "The PEIS-ecology project: a progress report," in *Proc. of the ICRA Workshop on Network Robot Systems*, 2007.

[40] L. Parker, K. Fregene, Y. Guo *et al.*, "Distributed heterogeneous sensing for outdoor multi-robot localization, mapping, and path planning," *Multi-Robot Systems: From Swarms to Intelligent Automata*, 2002.

[41] R. Arumugam, V. Enti, L. Bingbing *et al.*, "DAvinCi: A cloud computing framework for service robots," in *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, 2010.

[42] S. Zug, F. Penzlin, A. Dietrich *et al.*, "Are Laser Scanners Replaceable by Kinect Sensors in Robotic Applications?" in *IEEE Intl. Symp. on Robotic and Sensors Environments (ROSE)*, 2012.

[17] See an example at: www.youtube.com/watch?v=y6LqLNB4VDk