FH | JOANNEUM
Luftfahrt / Aviation

Bachelor's Thesis

Safety as Service

Service oriented Architectures for safety-critical Systems

Submitted by:	Stefan Lengauer
Registration number:	1210587029
Academic Assessor:	FH-Prog.Dipl.Ing.Dr. Holger Flühr
Date of Submission:	xx August 2015

Declaration of Academic Honesty

I hereby affirm in lieu of an oath that the present bachelor's thesis entitled

"Safety as Service - Service oriented Architectures for safety-critical Systems"

has been written by myself without the use of any other resources than those indicated, quoted and referenced.

Graz, xx August 2015

Stefan LENGAUER,

Preface

This thesis was written as result of the internship at VIRTUAL VEHICLE at Inffeldgasse, Graz from April to July 2015, which was conducted as part of the Degree Programme in Aviation at FH Joanneum in Graz, Austria.

The VIRTUAL VEHICLE research center is an international company, specialized in automotive and rails industry. In total, it has over 200 employees, which are splitted up in four main research areas. For the time of my employment, I worked as part of the Electrics/Electronics (E/E) & Software Area, ... more specificly .. functional safety.

During my employment is was part of the EMC2 project, a European project, ... my part soa

This thesis consists mainly of two documents, which were produced during my internship. The first one is a glossary, which aims at defining certain terms and unifying the opinions from different research areas. The second document, was an extensive investigation on how the service oriented architecture paradigm can be applied in a safety-critical embedded system. In detail, with a vehicle as the system.

Although the employment was oriented towards the automotive industry, the functional safety and fault tolerance concepts, reappear in a very similar way in other engineering disciplines, like aviation. Furthermore, the service oriented approach, which is considered in terms of automotive, will most likely become also an important issue for aviation in near future.

Furthermore, the location of the company at Inffeldgasse was another big advantage, for it allowed the conduction of various courses at the Technical University of Graz, alongside the employment.

At this point I want to thank my supervisor from FH JOANNEUM, FH-Prof. Dipl.Ing. Dr. Holger Flühr, as well as my supervisor provided by the company, Dipl.Ing. Helmut Martin. ...

and also Dipl.Ing.Dr. Andrea Leitner and Mr. Mario Driussi, which helped develop the ideas and concepts featured in this thesis during numerous meetings and discussions.

Contents

Abstract vii

Kurzfassung viii

List of Figures ix

List of Tables x

List of Abbreviations 1

1 Introduction 2

1.1 Related instances . . . 3

1.1.1 ISO 262662 international standard . . . 3

1.1.2 MISRA . . . 3

1.1.3 AUTOSAR . . . 3

1.2 service . . . 3

1.3 System . . . 3

1.3.1 Definitions . . . 3

1.3.2 Definition for the EMC2 project . . . 4

1.3.3 System of Systems . . . 5

1.3.4 System Layers . . . 6

1.3.5 Embedded System . . . 9

1.4 Component . . . 10

1.4.1 Definitions . . . 10

1.4.2 Component Interfaces . . . 12

1.5 Service . . . 17

1.6 Architecture . . . 17

1.7 Service oriented architecture . . . 17
1.7.1 Definition . . . 17
1.7.2 Historic development . . . 17
1.7.3 Structure of SoA . . . 17
1.7.4 Service composition . . . 17
1.8 Communication in SoA . . . 17
1.9 Dependability . . . 17
1.9.1 Reliability . . . 17
1.9.2 Availability . . . 17
1.10 Functional safety . . . 17
1.10.1 Safety related terminology . . . 17
1.10.2 Definition of safety . . . 17
1.10.3 Fault tolerance . . . 17

2 Methods **18**
2.1 SoA in Embedded Systems (Embedded SoA) . . . 18
2.1.1 Correlation of design philosophies . . . 18
2.1.2 Difference between Web SoA and Embedded SoA . . . 19
2.1.3 SoA in automotive . . . 23
2.2 Safety services . . . 25
2.2.1 Relation of Safety and security services . . . 25
2.2.2 Fault detection service (WDT) . . . 26
2.2.3 Error detection service . . . 27
2.2.4 Memory Protection Service . . . 27
2.2.5 Orchestration Management Service . . . 28
2.3 Use Case: Error Detection Service . . . 28
2.3.1 Service Investigation/Planning . . . 28
2.3.2 Service inventory analysis . . . 28
2.3.3 Service oriented analysis . . . 28
2.3.4 Service oriented design . . . 28

3 Results **30**

4 Discussion **31**

5 Conclusion **32**

Abstract

One of the major issues in the automotive industry is the constant growing complexity of E/E (Electrics/Electronics) systems and the thereof resulting fault propagation due to the strong interconnection of the systems. The area of *functional safety* is concerned with the prevention of non tolerable risks in event of error. This is conducted by identifying possible hazards, estimating the potential risks and developing necessary countermeasures, based on these investigations. The requirement therefore is an accurate and thorough comprehension of the observed E/E system.

The service oriented architecture is a design paradigm, which features the concept of software reuse by implementing functionalities as technology independent and loosely coupled services. Although the design paradigm is already standard for Web applications, it has not yet been applied in safety-critical embedded systems.

Thus, this Bachelor's thesis investigates the applicability of service oriented architectures for such systems, with a focus on *functional safety* and *fault tolerance* context.

The first part of the thesis is mainly a glossy, which defines certain terms, which are critical for the subsequent presented concepts. On this basis it is defined what *safety as a service* can mean in general, and how the actual implementation in a given safety-critical system may look like, in order to meant with the requirements specified in the ISO 26262 standard. This is the safety standard for safety-critical E/E systems in vehicles with a maximum gross weight of 3500 kg.

Kurzfassung

List of Figures

1.1 Relation of *system*, *component* and *element* according to ISO 26262 [6]. . . 5
1.2 The hierarchy of the integration levels of the system "vehicle" with examples 7
1.3 Relation of System, SoS and Service to one another [8]. 8
1.4 Implementation example of an ARROWHEAD framework [8]. 9
1.5 The interfaces of a component, with respect to the **GENESYS architecture** [2, p.40] 14
1.6 Illustration of the client-server communication [16]. 15
1.7 Illustration of the sender-receiver communication [16]. 15
1.8 Illustration of the different ports at a single component [16]. 16

2.1 Illustration of the overall system architecture by Gleirscher and Vogelsang [19]. 20
2.2 Relation of service oriented architecture to service oriented architecture in embedded systems. 22
2.3 Classification of various services into safety and security services. 25
2.4 Schematic design of a windowed watchdog timer [24]. 27
2.5 Schematic illustration of the working process of a *Sequenced Watchdog Timer* [24]. 28
2.6 Example architecture for an *fault detection service*, like a WDT. 29
2.7 Example architecture for an *error detection service*. 29

List of Tables

List of Abbreviations

Chapter 1

Introduction

Due to the very different comprehension of certain terms by different people from different fields of research The first part of my work during the internship was the creation of a glossary describing certain given terms ... in order to unify the understanding of these terms and create a basis for discussions ans researches.

- The glossary was later taken as standard for this tasks ... - and deals with the terms: *service*, *system*, *system of systems*, *architecture*, *service-oriented architecture*, *configuration*, *static reconfiguration*, *dynamic reconfiguration*, *inter-core communication*, *intra-core communication* and *binding*. The most important parts of this glossary will be covered within this chapter. - furthermore a database with the used literature was created in order to provide the necessary references for the glossary and provide a ... where to find further information for the employees.

- disambiguity safety and security - what is safety and fault tolerance

1.1 Related instances

1.1.1 ISO 262662 international standard

1.1.2 MISRA

1.1.3 AUTOSAR

1.2 service

1.3 System

This chapter explores various definitions of the term system, as well as related hierarchies and terminology. Because of its reappearance in later chapters, the term *embedded system* is also covered.

1.3.1 Definitions

Definition of *system* by different sources:

ISO/IEC 15288 . The ISO/IEC 15288[1] standard states that systems *"are man-made and may be configured with one or more of the following: hardware, software, data, humans, processes (e.g. processes for providing services to users), procedures (e.g. operator instructions), facilities, materials and naturally occurring entities"* [1].

GENESYS architecture . Definition of system according to Obermaisser and Kopetz: *"an entity that is capable of interacting with its environment and is sensitive to the progression of time"* [2, p.7]. The *environment* is a system itself, which produces input for other systems and acts according to their outputs. Which elements (cf. section 1.3.1) belong to the system, and which to the environment, is a matter of perspective.

ISO 26262 . The ISO 26262 standard defines a system as a *"set of elements that relates at least a sensor, a controller and an actuator with one another"* [3]. This definitions is already a bit more specific with a focus on the automotive industry.

[1]The ISO/IEC 15288 standard is a *System Engineering Standard* and defines processes and terminology for systems created by humans [1].

ARROWHEAD . Arrowhead has a bit slightly different notion of the concept *system*. With respect to their view, a system is simply an artifact which provides and/or consumes *services* (cf. chapter **??**) [4], what implies that systems are directly related to service oriented architectures (cf. chapter **??**).

AUTOSAR .

> *"An integrated composite that consists of one or more of the processes, hardware, software, facilities and people, that provides a capability to satisfy a stated need or objective"* [5].

There is no indication that system have any internal structure, or that services are exchanged inside of them, which is the case according to the other sources. In this definition the is more like a component (cf. *"It may also be referred to as Component or Device"* [4]).

The sources may differ a bit in terms of specificity, but all in all, there are no critical contradictions or discordances.

Throughout this glossary the naming convention of the ISO 26262 standard, which can be seen in figure 1.1 is used.

System Element

> *"element - system or part of system including components hardware, software, hardware parts, and software units"*, [3]

This definition of *element* is taken from the ISO 26262 standard. It is a very generic term and thus it is not advisable to define it for any entities at a specific layer or with a specific characteristic. Instead, according to the quote, it can be more or less any entity of a system, since a system itself is defined as "set of elements" [3].

1.3.2 Definition for the EMC2 project

The most crucial aspects are combined and emphasised in the following definition.

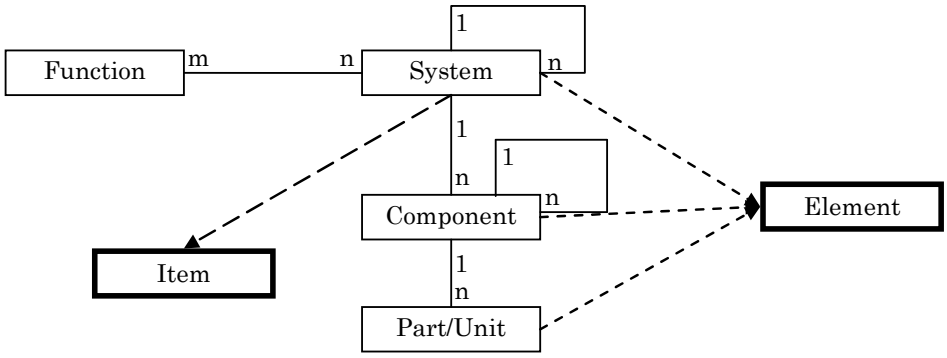


Figure 1.1: Relation of *system*, *component* and *element* according to ISO 26262 [6].

A system is a hierarchical composed, time sensitive element, which interacts with the environment by processing input and providing output in turn.

It is concerned with satisfying a specific need or purpose and disposes of a, more or less complex, internal structure, which may include hardware, software and data.

As stated, the time domain has to be taken into account to describe the behaviour of a system. Thus, a time model is required. Distributed systems usually deploy physical clocks for splitting the time in equally spaced intervals [2, p.7-8].

For the scope of this paper, the overall system is defined as a **vehicle** in order to be able to provide some examples. The environment consists therefore of other vehicles or the surrounding infrastructure. Nevertheless, the entire traffic could also be taken as a system and the environment would then be a different one; for example entities like roads, weather conditions and the like.

1.3.3 System of Systems

Systems are hierarchical and can be composed or decomposed into sets of interacting constituting systems. Often this is referred to by the term *System of Systems (SoS)* [2, p.7]. Thus, the SoS is in general the level above a given system and is therefore dependent on the definition of the related systems. With respect to our previous example of a system vehicle, a SoS could be for example the traffic of Graz, with many vehicles participating.

This is in accordance with the definition of system by Arrowhead: *"collaborating, compliant systems become system-of-systems and SoSs can be parts of other, bigger SoSs in turn"*

[4].

System of systems features the following five characteristics:

- operational independence of its systems,
- management independence of the systems,
- evolutionary development,
- emergent behaviour, and
- geographic distribution [4].

Concerning this definition, it has to be stated that *geographic distribution* it not necessarily always the case when operating with embedded systems.

1.3.4 System Layers

The parts of a system can be divided into different layers of implementation, for this kind of abstraction makes it easier for human beings to comprehend the overall relations. Unfortunately each field of research features its own way of fractionising systems and most of the approaches are hardly hardly compatible or even contradicting. The following, a segmentation from the GENESYS architecture, revealing a quite hardware and embedded systems oriented point of view, and a more abstract view from the ARROWHEAD project, are investigated in more detail.

System Layers by GENESYS

The GENESYS architecture by Obermaisser and Kopetz distinguish between three different layers, denoted *chip-level*, *device-level* and *system-level* [2, p.44]. An example of the hardware elements at different levels by means of a system vehicle can be seen in figure 1.2.

System Level . The system level consists of *Devices*, which are themselves logically self-contained apparatus. With respect to a system vehicle this could be for example an ECU, a sensor, an actuator or the like [2, p.45].

Device Level . The devices at the system level, contain a certain internal structures themselves. In case of embedded systems these are in most cases *Chips* [2, p.45], like the

the AURIX™ [2] chip, which is frequently used in the automotive industry.

Chip Level . According to the implementation layers in the GENESYS architecture, the chip level is the lowest level of implementation. In case of an MPSoC (Multiprocessor System-on-Chip)[3] this level contains the single IP Cores of the chip [2, p.46]

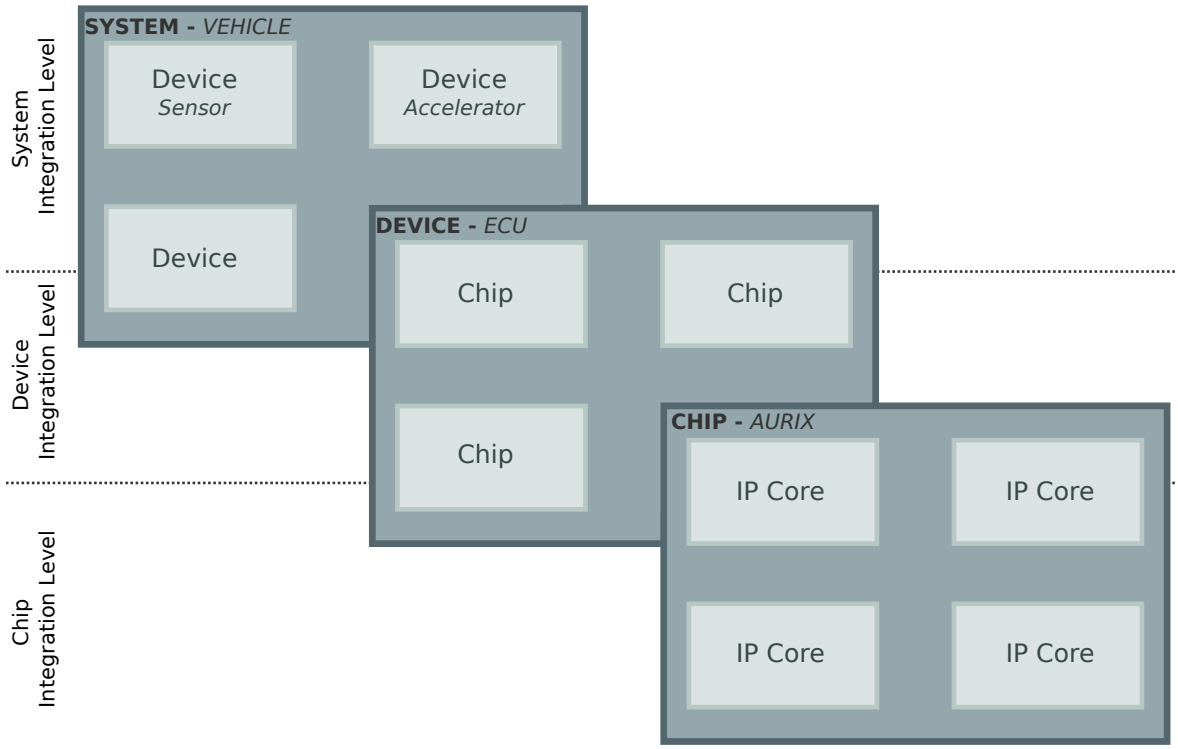


Figure 1.2: The hierarchy of the integration levels of the system "vehicle" with examples

System Layers by ARROWHEAD

The Arrowhead Framework provides an hierarchy by means of documentation documents, which are spitted in the three levels *system-of-systems*, *system* and *service* [4]. Their involved documents and relations are pictured in figure 1.3.

System-of-systems . The levels features the two documents *SoS Description (SoSD)* and *SoS Design Description (SoSDD)*. The differ in the amount of information they are

[2]The AURIX (Automotive Realtime Integrated NeXt Generation Architecture) is a new family if multi core microcontroller, adjusted to the needs of the automotive industry - `http://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-tm-microcontroller/aurix-tm-family/channel.html?channel=db3a30433727a44301372b2eefbb48d9`

[3]A *Multiprocessor System-on-Chip* is an integrated circuit which uses multiple processors and is often used for embedded applications [7].

revealing. While the first represents only an abstract view, the second also reveals the implementation of the SoS and its technologies [4].

System . The system level comes with the two documents *System Description (SysD)* and *System Design Description (SysDD)*. Just as at the SoS level the first one features a kind of black box opacity and the second a white box view (cf. section **??**) [4].

Service . The service level contains the four documents *Service Description (SD)*, *Interface Design Description (IDD)*, *Communication Profile (CP)* and *Semantic Profile (SP)*. The service description is referred to in **??**.

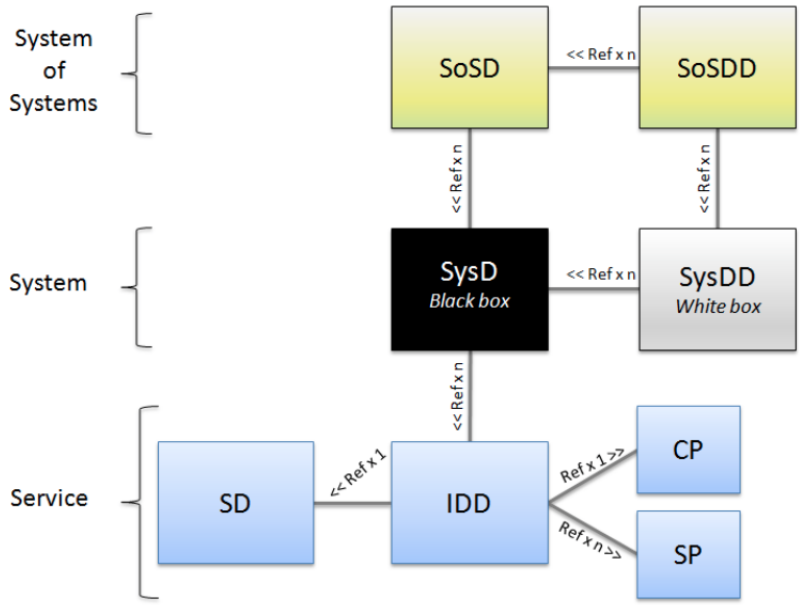


Figure 1.3: Relation of System, SoS and Service to one another [8].

All the documents exist as templates which should be filled out during the development process of a system. They feature a XML like style in order to be human and machine readable at the same time.

Developed systems can then be constituted to SoS by an underlying cloud, as depicted in figure 1.4. All the system have specified and standardised interfaces and work together by means of three *core services*, labeled II, AI and SM.

Information Assurance (IA) . This service is responsible for providing secure information exchange through authorization and authentication [8].

Information Infrastructure (II) . The II service enables the listing of the services in the *service repository* (cf. section **??**) and their discoverability [8].

System Management (SM) . This is the core service for the system of systems composition and features logging and monitoring abilities [8].

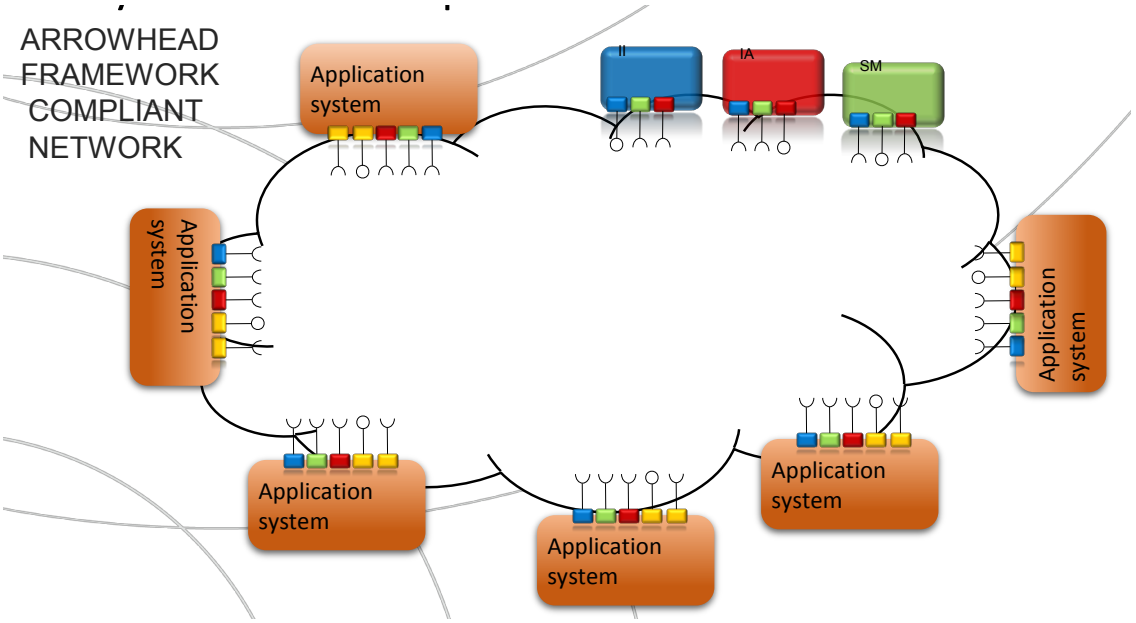


Figure 1.4: Implementation example of an ARROWHEAD framework [8].

1.3.5 Embedded System

Embedded systems are computational modules integrated to physical devices and equipment. The have a predefined set of tasks and requirements and are capable of processing information [9] [10, p.xiii].

Compared to general-purpose computation systems they usually dispose of less processing resources and come with narrower operation ranges, but at the same time they feature a high efficiency by optimally managing the available resources [11, p.283] [10, p.5]. Also, their presence is usually quite unobtrusive, because instead of mice and keyboard the user interface consists of typical input devices like buttons, steering wheel, or pedals.

Embedded systems are *reactive systems*, what means that they perform a continuous interaction with the environment. The connection to the physical environment is realised by means of sensors, responsible for collecting information, and actuators for performing the actual reaction [10, p.8-9].

During operation, embedded systems are in a certain state, waiting for input. When provided with that, they perform computations and generate an output, which is handed back to the environment [10, p.9].

Embedded systems in safety-critical applications have to care about the issues *time constraints*, *dependability* and *efficiency requirements*.

Time constraints . One challenge of Embedded Systems is the meeting of so called *time constraints*, which basically means the conduction of a computation within a specified time [10, p.8-9] [9]. Kopetz states *"A time-constraint is called hard if not meeting that constraint could result in a catastrophe"* [12].

Dependability . Embedded systems, operating in safety-critical environment, like nuclear power plants, cars, trains or aircraft, must be *dependable*, for they are directly connected to the environment and have immediate impact on it. The Dependability is split up in further aspects - in detail *Reliability* (cf. section **??**), *Maintainability*, *Availability* (cf. section **??**), *Safety* (cf. chapter **??**) and *Security* [10, p.4-5].

Efficiency requirements . Efficiency is a key concept of embedded systems and is concerned with providing a maximum computation performance while minimizing the required energy. The efficiency is measured in "operations per Joule" and has been been increasing almost exponentially during the last twenty years.

1.4 Component

For the EMC2 project it is crucial to achieve a clear distinction between the term *component* and *service* (cf. chapter **??**). Due to the historic development of *service oriented architecture* (cf. chapter **??**), as successor of the *component based software engineering* (cf. chapter **??**), those terms have often been put on a level, giving rise to confusion.

Another issue is the term *software component*, which leads to the question, whether components are pure software elements.

In the following some definitions from different relevant resources are presented and discussed, with the aim to eliminate those ambiguities. The relation of component to service is investigated in section **??**, subsequent to the definition of service.

1.4.1 Definitions

GENESYS reference architecture . Obermaisser and Kopetz state that a component is a software or hardware unit that performs a specified computation within a given period of

time [2, p.38] and communicates with other components by means of specific *interfaces* (cf. section 1.4.2). Like systems, components are hierarchical and therefore dependent on the point of view - from a different viewpoint a quantity of components may be seen as a single component. With respect to this definition all entities in figure 1.2 (*System*, *Sensor*, *AURIX*, etc.) can be denoted as components.

ISO26262 . The ISO 26262 standard supports this concept by its definition of a component as a *"non-system level element that is logically and technically separable and is comprised of more than one hardware part or one or more software units"* [3].

Software Component

The following sources refer explicitly to *software components*.

Szyperski . Szyperski features a definition with emphasis on the software oriented point of view: *"Software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system"* [13, p.xxi].

AUTOSAR .

> *"Software-Components are architectural elements that provide and/or require interfaces and are connected to each other through the Virtual Function Bus to fulfill architectural responsibilities"* [5].

AUTOSAR defines a *Software Component* as encapsulation for parts of the automotive functionality, but it dictates no specific "granularity", meaning that an AUTOSAR software component might be either a *"a small, reusable piece of functionality (such as a filter) or a larger block encapsulating an entire sybsystem. [14]"*

Software Engineering Institute . In software engineering a component is usually considered as a piece of software with specific functionality. The *Software Engineering Institute*[4] describes a component by means of another definition by Szyperski:

> *"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can*

[4] *"The Software Engineering Institute (SEI) is a non-for-profit Federally Funded Research and Development Center (FFRDC) at Carnegie Mellon University, specifically established by the U.S. Department of Defense (DoD) to focus on software and cybersecurity."* - `http://www.sei.cmu.edu/about/organisation/index.cfm`

be deployed independently and is subject to composition by third parties" [13].

ARCITURA [5]. *"A component is a unit of logic that exists as a standalone software program as part of a distributed computing architecture"* [15].

Definition for the EMC2 project

A component is a logical and technical separable hardware or software unit that is capable of performing a specific computation.

It offers an abstraction that simplifies the understanding of complex systems. Therefore they have to be hierarchical, meaning that components can be composed to other, larger, components.

As suggested by this definition the term component should be used for both, software hardware. If not denoted specifically as *software component* or *hardware component*, the term can be referred to both throughout this glossary.

A component can be seen as *black box* (cf. chapter **??**), meaning that the more or less complex internal structure is invisible or not of concern for the user. Therefore, other components stay unaffected from modifications of this internal structure, given that the behaviour at the *Linking Interface* (cf. section 1.4.2) remains unaffected [2, p.38-39] [16] [17].

As a self-contained subsystem, it can be developed and tested independently, and latter be used as building block for systems or higher level components. In other words, the components are the basic building blocks of a system [18].

Nevertheless, not everything is a component. According to Sametinger, an algorithm in a book is not a component, but it has to be implemented by means of an arbitrary programming language and equipped with well-defined interfaces in order to become a component [17, p.2-3].

[5] Arcitura™ Education Inc. is a global provider of progressive, vendor-neutral training and certification programs. It is also the owner of `http://www.serviceorientation.com` [15], which served as valuable sources and is also the located of the related citation - `http://arcitura.com/about`

1.4.2 Component Interfaces

The interfaces are necessary for any interaction with other components. In the following two different approaches for describing these interfaces are presented. The first one is from the **GENESYS project** and features a quite implementation oriented point of view. The second definition, by **AUTOSAR**, is more abstract and better adjusted to the service oriented architecture paradigm.

Interfaces by GENESYS

Following the definition from Obermaisser and Kopetz, each component may dispose of up to four interfaces for communication with other entities. The *Linking Interface*, the *Local Interfaces* and the *Technology Independent-* or *Technology Dependent Interface*. They are illustrated in figure 1.5 [2, p.40-41].

Linking Interface (LIF) . The *Linking Interface* is a message based interface and responsible for offering the component's services. The LIF is dependent on the level of integration, e.g. Inter-IP Core LIF at chip level or Inter-Chip LIF at device level. Nevertheless, it is used only for communication to other components at the same layer and also the only place, where a component may provide its services to other components [2, p.9].

The *Linking Interfaces* are always technology agnostic, which means that they do not expose details on their implementation or *Local Interfaces*. Accordingly, the implementation can be modified, without other components noticing, as long as the specification at the LIF remains unchanged [2, p.9, 40-41].

Local Interface . The *Local Interfaces* establish the connection between a component and its local environment, which could consist of sensors, actuators and the like. If the environment is modified, the semantics and timing of the data should stay the same in order to do not violate the specification. A *Local Interface* could also be mapped to a *LIF* of a component at the next-higher level - this is known *gateway component* (cf. chapter **??**) and enables different layers to communicate with each other.

Components do not necessarily require local interfaces. Such are denoted *Closed Components* [2, p.40-41].

Technology Independent Interface (TII) . The *TII* is the instrument for configuring and reconfiguring a component, e.g. assigning a name, configuring input and output ports or

monitoring the resource management. Starting, restarting and resetting the component is also executed through this interface. The TII communicates with the hardware, the operating system and the middleware, but not with the *application software (service)*, which is reserved for the LIF [2, p.40-41].

Technology Dependent Interface (TDI) . The *Technology Dependent Interface* enables a look inside the component and allows to inspect internal variables and processes. Thus, it is reserved for people who bring a deep understanding of the components internals and is of no relevance for the user of the LIF services [2, p.40-41].

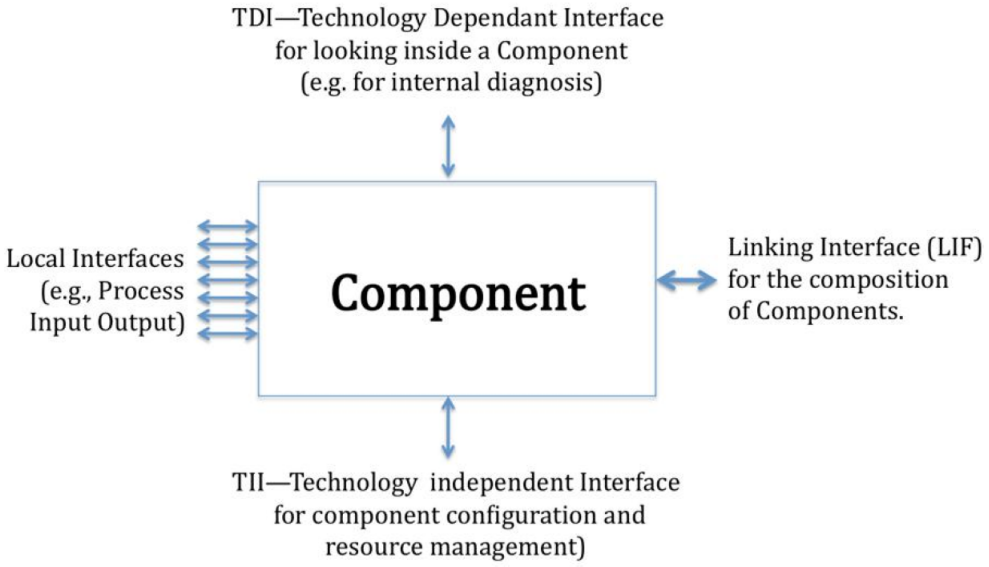


Figure 1.5: The interfaces of a component, with respect to the **GENESYS architecture** [2, p.40]

Interfaces by AUTOSAR

AUOSAR follows a slightly more abstract approach concerning the interfaces of a component. According to their definition, a component may dispose of a number of *ports*. A port belongs to one component only and is the interface a component uses to communicate with other components.

Within this context, the term *interface* specifies a kind of contract or specification, on which services can be called at this port, and the format of the data emitted at this port.

There are four different types of interfaces, belonging to the two different communication patterns **Client-Server** and **Sender-Receiver**: [16]

Client-Server Communication . When this kind of communication is performed the *client-component* requests the a specific service from the *server-component* and sends neces-

sary parameters. The server then processes the incoming request and returns a response. A single component can be a server and a client at the same time [16]. A schematic illustration of this type of communication is pictured in figure 1.6, where SW-C denotes *software component*.

Sender-Receiver Communication . The *Sender-Receiver* approach is a bit different. The task of the sender is to distribute his information to one or more receivers, without ever getting a response in form of data or control flow. In fact he does not even know the number or identity of the receivers. Those have to decide on themselves, how to deal with the received data (cf. figure 1.7) [16].

The different ports and their properties are visualized in figure 1.8.

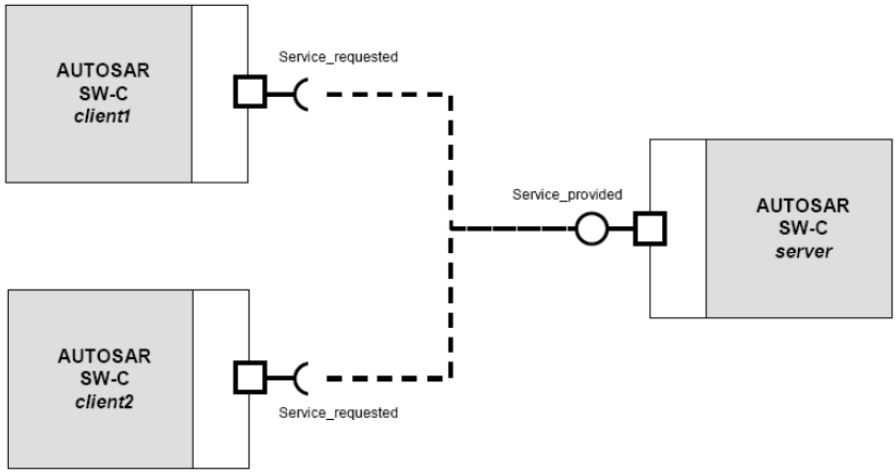


Figure 1.6: Illustration of the client-server communication [16].

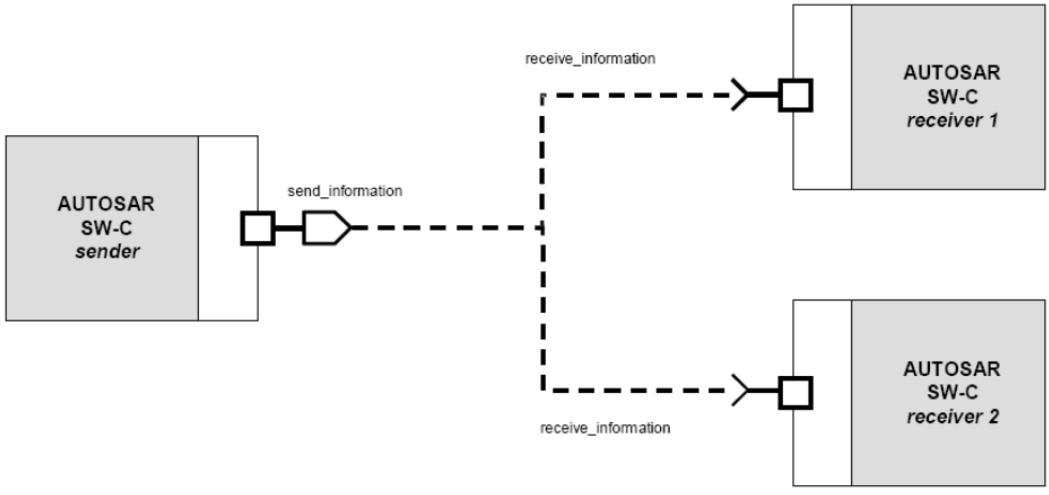


Figure 1.7: Illustration of the sender-receiver communication [16].

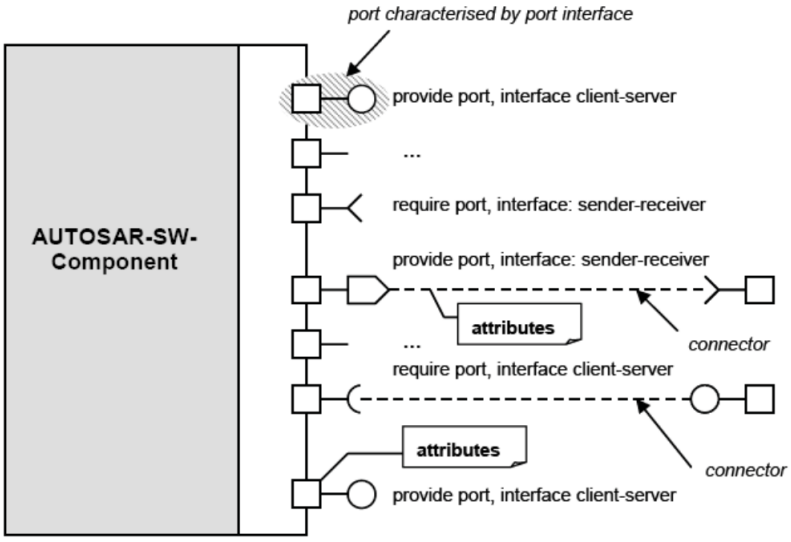


Figure 1.8: Illustration of the different ports at a single component [16].

1.5 Service

1.6 Architecture

1.7 Service oriented architecture

1.7.1 Definition

1.7.2 Historic development

1.7.3 Structure of SoA

1.7.4 Service composition

1.8 Communication in SoA

1.9 Dependability

1.9.1 Reliability

1.9.2 Availability

1.10 Functional safety

1.10.1 Safety related terminology

1.10.2 Definition of safety

1.10.3 Fault tolerance

Chapter 2

Methods

This chapter is structured in the three main section, labeled *SoA in Embedded System (Embedded SoA)*, *Safety services* and *Use Case: Error Detection Service*.

The first section deals extensively with the possible application of the service oriented design paradigm in embedded system and deals with the challenges this approach has to face when approached with safety-critical requirements. Moreover, the differences to conventional SoAs, as Web application is investigated in detail. The second section presents xx services, which are relevant for *functional safety* in detail and presents their intended functionality, as well as possible implementations in terms of an architecture.

The final part consists of a simplified use case, dealing with the design of an error detection service with respect to the design phases *service investigation/planning*, *service inventory analysis*, *service oriented analysis* and *service oriented design*.

Most of the findings and concepts in this chapter is "not yet" covered in literature, but is the result of numerous meetings and discussions.

2.1 SoA in Embedded Systems (Embedded SoA)

2.1.1 Correlation of design philosophies

The CBSE and SoA are coexisting design philosophies and usually either one or the other is applied in a given system. In contrast, Gleirscher and Vogelsang present in [19] a proposal, how both paradigms can exist in parallel in a single system.

The presented system models consists of two layers with three different models. The *feature layer* contains the *feature hierarchy* model, which describes all the features a vehicle

presents to its user. The *platform layer*, in contrast, is the technical realization and consists of the *service hierarchy* and the *component model* [19].

This relation is depicted in figure 2.1.

Feature Hierarchy . Alongside *services* and *components* their model disposes of a third hierarchy, denoted *features*.

According to [19], a feature "is a behavioral property of a system observable (and influenceable) by its user." AUTOSAR describes it by the following quote:

> *"The term feature is commonly used in the software tool community to describe characteristics (functionality) of the software"* [5].

The *feature* has a many to many relationship to the services, since a service could offer various features, but one and the same feature, can also be offered by the very same service. For example the service *braking service* could contain a *safety braking mechanism*, which is responsible for performing an emergency brake when needed [19].

The set of features for a specific vehicle is given by its configuration in terms of hardware software, etc. A convertible car may include a feature for automatically opening and closing the top, while a sports car could allow to switch to a "sport mode."

Service Hierarchy . The services are the link between the features and the technical platform, they are realized on. A service can be provided either directly by a "material component", like a sensor or an actuator, or it can be composed out of various other services [19]. For an acceleration sensor, the corresponding service would be the *acceleration measurement*.

Component Model . The components are the material or immaterial, in other words hardware and software, parts of a system. Accordingly, a material part is for example a sensor and an immaterial part a communication software.

2.1.2 Difference between Web SoA and Embedded SoA

The service oriented design paradigm was originally designed for the application in the web, which offered, more or less, ideal prerequisites for this kind of architectural style. There is an underlying network already available, which allows interconnection, and *time constraints* are

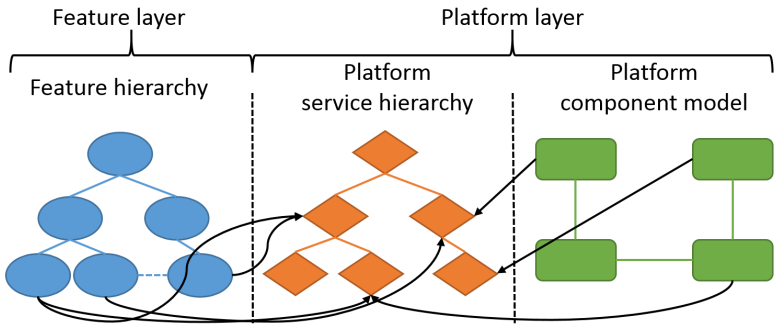


Figure 2.1: Illustration of the overall system architecture by Gleirscher and Vogelsang [19].

quite ..., because delays are unlikely to cause a disaster. Thus, it is not surprising that web services are the application area, where SoA has scored the highest market penetration [9] [20].

In contrast to web services, *embedded systems* are networks consisting of numerous interconnected nodes, with diverse measurement-, steering-, or computation capabilities. They have to face certain challenges like *limited resources*, *different complexity of hardware* [21] [22].

Some of the issues, the SoA has to face in embedded systems are described in the following.

Limited resources . One obvious major drawback are the very limited resources in embedded systems, which are designed for highly specialized purposes and lack computation power as well as storage size [9] [21] [22].

Also the set of components is determined at assembly time ...

Different levels of complexity . The complexity usually varies greatly. The applied hardware components may include very simple sensors, with low resources and capabilities as well as very advanced nodes like MPSoCs) [21] [22].

Accordingly, there are high level information systems services as well as low level, generic embedded system services. SoA has to deal with the connection and integration of them [9]. Especially when it comes to the interaction of SoAs in embedded systems with SoAs at higher levels.

Event- and data-driven . In contrast to web services, an embedded system disposes of (many) sensors which are interconnected in a network. Thus, the ad-hoc "request-response message pattern" which is known from web services does not work, for embedded systems are event- and data driven: For example a sensor measures some data

and publishes it to all connected services. Those can then act according to the received input, e.g. trigger actuators or other services. This is denoted as "fire and forget" scheme [22].

Lifespan of services . Another difference to web services is the lifespan of services. While web services are used to work only a limited number of hours (or even minutes), the services in embedded systems could have application times of multiple years, e.g. a measurement service for a sensor [20].

Dynamic character . This is even aggravated by the dynamic character of embedded networks - *"new nodes may enter the network, existing nodes may fail and network characteristics can change over time, especially if wireless communication media are used"* [22].

Time constraints . Embedded systems are time-critical, meaning that computation must be conducted within a given time window in order to allow the correct operation of the system. Especially in a safety-critical system like a vehicle, which is used to operate at high velocities, a violation of those time constraints could cause serious incidents.

These obstacles give reason to the statement that web services and safety-critical embedded systems are non-related areas [9].

Nevertheless, there are a lot of benefits in applying a service oriented architecture style, like decoupling configuration from environment, improvement of reuseability, maintainability, higher level of abstraction, interoperability, more interactive interfaces between devices and information systems.

[20].

The SoA paradigm offers a promising possibility to overcome many of embedded systems related drawbacks and implementing some of the stated advantages [20] [22]. Thus, it is not surprising that there are various different projects dedicated to investigate the applicability of SoA for embedded systems. Those include SIRENA, SOCRATES, OASiS, MORE, RUNES and εSOA. In the proceeding the εSOA project by Scholz, Sommer et al. [21] [22] [20] will be investigated in more detail.

eSoA

The εSOA project from Sommer, Gaponova and Buckl from the Department of Informatics at TU München, is an approach to master the obstacles which come with applying service oriented architectures in embedded systems.

However, the proposed architecture omits some functionality which is included in conventional SoAs, like the dynamic detection of services within the network. Moreover it features *long term service composition* and uses external tools for the reconfiguration. However, a necessary requirement is the ability to update deployed services and migrate services to new hardware without a complete system interrupt [22].

Embedded SoA

Unfortunately, there are not many sources covering the topic of SoA in embedded systems yet. Accordingly, the term *embedded SoA* does not appear in literature, but has been defined during the EMC2 project, in order to refer to service oriented architectures in embedded systems. The relation of "embedded SoA" to "conventional SoA" has been defined as it can be seen in figure 2.2. According to these findings the *embedded SoA* operates at a lower, very hardware oriented level, with a predefined and unchanging component architecture.

The SoA is located "above" embedded SoA and connected by interfaces. With regard to the example of a vehicle, the embedded SoA operates on elements like a power train, various devices (sensors, actuators, ...) etc, while SoA level contains the vehicle as overall system, but also other vehicles, which can be referred to as the *System of Systems*.

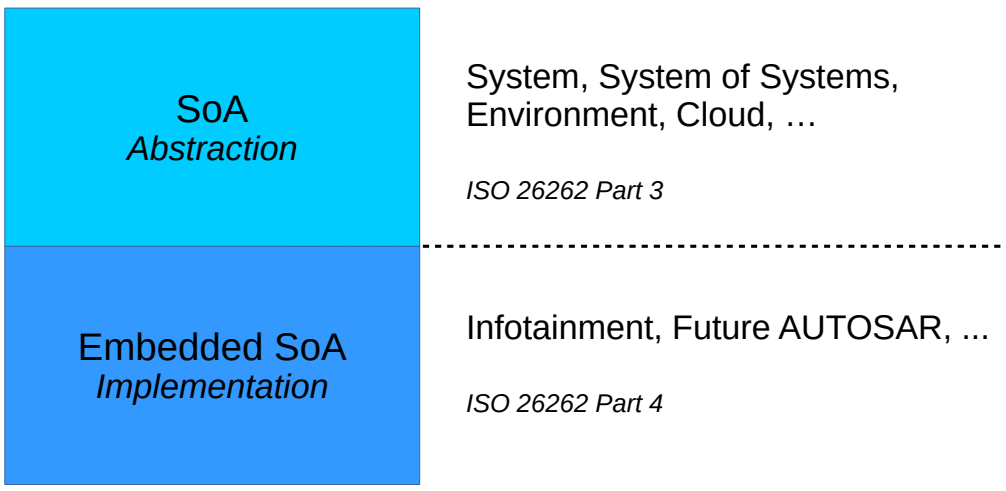


Figure 2.2: Relation of service oriented architecture to service oriented architecture in embedded systems.

2.1.3 SoA in automotive

An important subpart of the EMC2 project is the investigation of the applicability of the SoA concepts for the automotive industry.

In the future of automotive all vehicles will be, most likely, interconnected with each other and also their environment. This could give them opportunities like automatically detecting whether the traffic lights are green at a crossing and acting automatically. Within this document those futuristic vehicles, which are interconnected with their environment, are referred to as "connected cars".

The opportunities and advantages of the SoA approach have been dealt with in the previous section. With respect to vehicles its major advantage would be the possibility to reduce the quantity of computation hardware, because at the present time each component disposes of his own dedicated hardware for this task. Many of those are unused for the majority of time, but nevertheless they come with a lot of extra weight. As stated in [10, p.7], "Embedded systems should exploit the available hardware architecture as much as possible."

The SoA approach might reduce not only the weight, but also the complexity and costs of the overall vehicle. So far it is usually the case that every vendor used its own proprietary network and additional hardware, prohibiting the application of hardware components from another vendor [22].

Despite of all these advantages, right now the SoA paradigm is hardly applicable for vehicles. On the one hand, this is due to the strict regulations which apply for safety critical real-time systems like vehicles, which do not yet include any requirements with respect to SoAs. In terms of automotive the safety aspect also has to be considered, since vehicles are safety-critical systems where an error or a malfunction can easily harm people [23].

On the other hand, there are technical constraints. The AUTOSAR architecture, which is widely applied today, is not constructed for dynamic reconfiguration or binding of services, but everything has to be specified at building time. Still, there is already an approach by AUTOSAR, denoted "Future AUTOSAR", which is dealing with this very issue.

Although, the SoA for vehicles is still a long way ahead, it might be easier applicable at other levels, like the *System of Systems* level, meaning other vehicles and the overall traffic. For self-driving vehicles there could be for example services for the communication with traffic lights. If the the traffic lights in Germany would use another service as the ones in Austria, a SoA would allow to dynamically bind the new service when the border is approached. There are already projects which are dealing with the communication at this level of implementation.

Location of the service repository

The service repository is described in detail in section **??**, but its exact location and implementation in connection with automotive remains ambigous.

For SoAs in automotive it is likely that there exist various distributed repositories. One repository inside of every vehicle, containing all the services provided by the vehicle itself, which is also accessible if the vehicle is operating in urban regions and is not able to connect to its environment. For the SoA inside the vehicle, there is no other possible location, because otherwise it would be operable only as long as it is connected to the Internet, or whatever platform, which is used for the interconnection with the environment.

For the scope of this document these repositories will be referred to by the term "local service repository". Examples for the services they hold are services which origin from hardware components of the vehicle like, acceleration measurement, temperature measurement, communication services and the like. But also safety critical services for fault- and error detection, which need to be available whenever the vehicle is operated.

The counterpart to the "local service repository" is denoted by "external service repository". This could also be physically distributed and holds mostly services which are needed for interacting with the environment. For "self-driving"- or "connected cars" it could host all the services for managing the traffic. With an example: A traffic light in Graz publishes its service, "Show me the traffic light signal", somewhere on a server, where it is detected and bound by a vehicle driving in Graz, which is then able to decide whether it has to stop at an intersection.

Other services which could be located in this repository are *update services*. If an update for an existing service in the local repository is available, the vehicle could detect this automatically and subsequently load and install the specific service.

Service Contract

As already specified in section **??**, the service contract is the complete and extensive description of the service and should contain, with regard to vehicles, Documents from AUTOSAR, the ISO 26262 standard and also documentation regarding *functional safety*. It is more or less the goal of work package 1 of the EMC2 project to extend the *service contract* by a functional safety part.

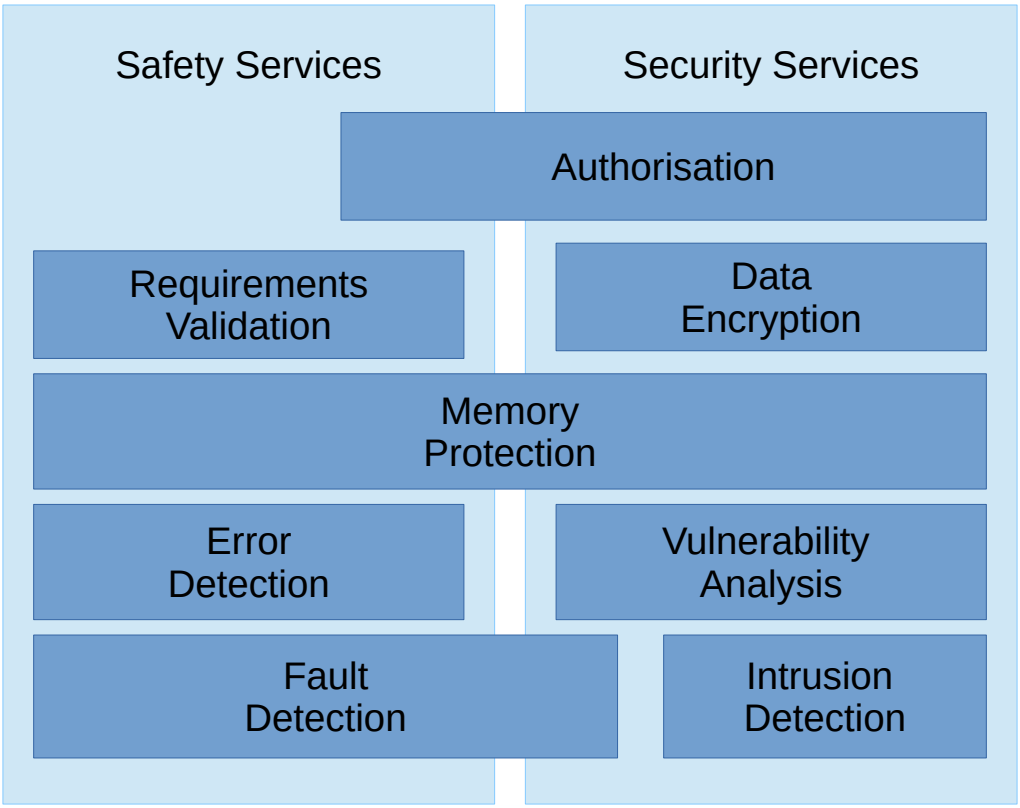
At the beginning of a service development process the contract is just an empty template

which gets filled more and more as the development advances. A template, which could be used for the development of services in the automotive industry is already provided by the ARROWHEAD project.

2.2 Safety services

2.2.1 Relation of Safety and security services

Figure 2.3: Classification of various services into safety and security services.



The most frequent application of service oriented architectures is in the field of Web services, where safety does not play a major role. However, in recent times this philosophy is also emerging in safety critical embedded systems, where specific safety and fault tolerance requirements have to be met.

There are many challenges for safety and security management in SoA systems, like the distributed hardware and software structure, loosely coupled components from different vendors, etc.

2.2.2 Fault detection service (WDT)

The fault detection service is described by means of its most prominent "Vertreter", the so called Watchdog Timer.

The *Watchdog Timer (WDT)* is an fault detection service, which is, dependent from its implementation, also capable of detecting *control flow errors*. Technically, the WDT is a simple timer circuit with a specified threshold time. If this limit is reached, it changes its state, which triggers further actions, like restarting a component or activating another safety service [24].

The advantages of the WDT is the simple design, which reduces the additional complexity of the overall system, as well as the costs. Concerning the functional principle, there are three slightly different designs, with individual capabilities and advantages [24].

Standard Watchdog Timer . With this basic setup, the service mirrored by the WDT periodically sends a simple heartbeat signal, indicating that the service is alive and active. This signal resets the timer. It a predefined amount of time elapsed, without an incoming signal, it is assumed that a fault has occurred and the WDT changes its state [24].

In terms of *control flow errors* in a service, the execution sequence of the instructions of a service is changed, which means that the heartbeat signal may, or many not, be sent too late. Thus, in the latter case, the WDT is capable of detecting the error [24]. The probability of noticing such an error is higher the closer the threshold time is to the time between the heartbeat signals.

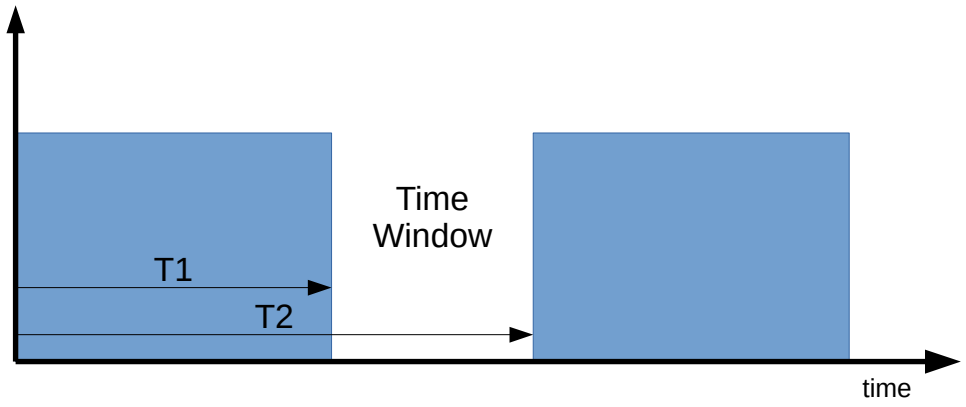
Windowed Watchdog Timer . The *Windowed Watchdog Timer* is an improvement of the standard version, which is capable of detecting most of the *control flow errors*. This is enabled by the application of two timers instead of one. With two timers the WDT is able to specify a time window, during which the heartbeat signal from the mirrored service must be received. The WDT is triggered if it receives a signal outside the window, or the timer reaches its threshold [24]. This is illustrated in figure 2.4 with the $time$ on the x-axis and the Timers labeled $T1$ and $T2$.

In case of an *control flow error* this signal is a bit ahead or past in time. The error detection coverage increases with narrowing the time window [24].

The advantage of this design is clearly that it allows the detection of more errors, but

on the other hand the implementation is slightly more complex.

Figure 2.4: Schematic design of a windowed watchdog timer [24].



Sequenced Watchdog Timer . This design is a further improvement of the *Windowed Watchdog Timer* and bases on the same principle. In contrast, to the other designs, the signal sent from the mirrored service carries a sequenced parameter. Only if the signal arrives in time and within the specified time window, this parameter is evaluated and compared to a parameter inside the WDT. If those match the sequence variable in the WDT is incremented and the timer reseted, starting the cycle over again [24]. The whole proce ss is illustrated in figure 2.5.

The disadvantage of this design is the clearly higher complexity, for the *Sequenced Watchdog Timer* must be capable of holding and modifying state information as well as comparing it to received information.

2.2.3 Error detection service

2.2.4 Memory Protection Service

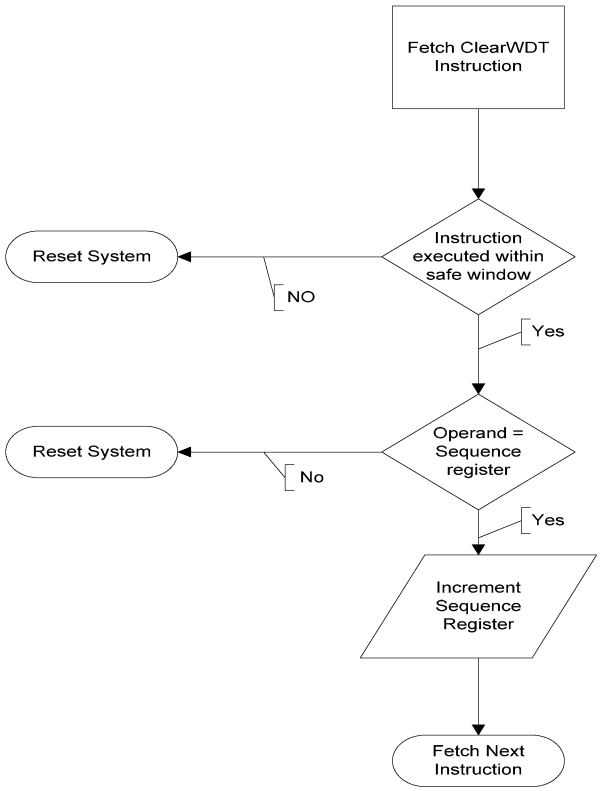
Memory protection is related to *safety* as well as *security*. It is a method for preventing processes or users from accessing memory that is not allocated to them.

Former embedded systems, or such that are small in size, do not necessarily require memory protection mechanisms, but modern, large scale system, have several advantages ... and with the increase of computing power the overhead is no longer crucial [25].

There are several aspects that stress the need for memory protection in embedded systems:

- It can serve as fault and error detection and containment mechanism, preventing a failure of one service to propagate and infecting the whole system [25].

Figure 2.5: Schematic illustration of the working process of a *Sequenced Watchdog Timer* [24].



- It is an important aid in the development process and helps at debugging by identifying "illegal" behaviour of erroneous services [25].
- It is also crucial from a security point of view, because it prohibits unauthorised access.

It is bizarre that memory protection is nowhere mentioned in connection with service oriented architectures

2.2.5 Orchestration Management Service

2.3 Use Case: Error Detection Service

2.3.1 Service Investigation/Planning

2.3.2 Service inventory analysis

2.3.3 Service oriented analysis

2.3.4 Service oriented design

Figure 2.6: Example architecture for an *fault detection service*, like a WDT.

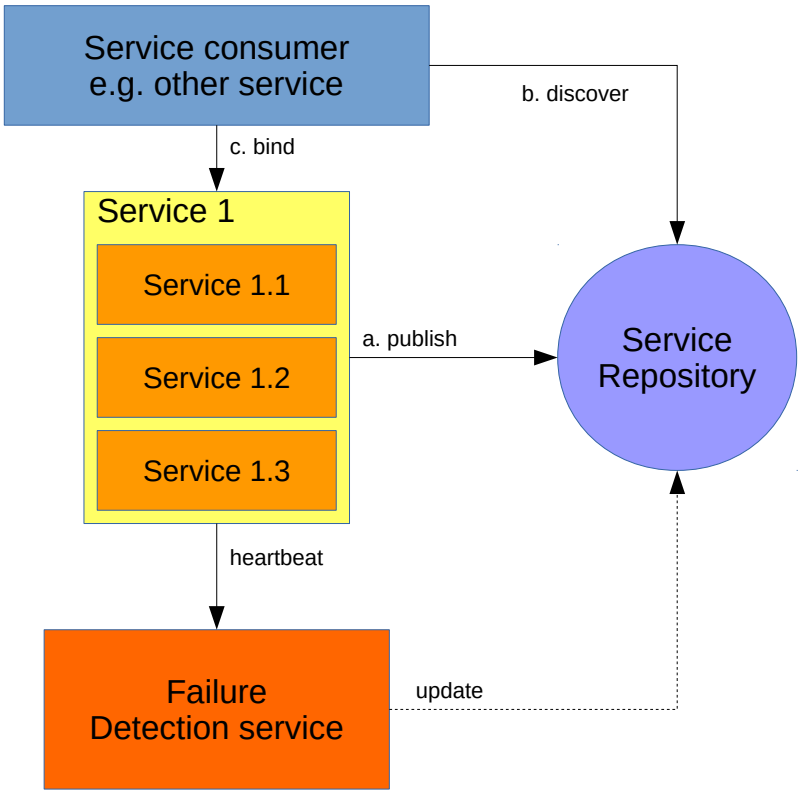
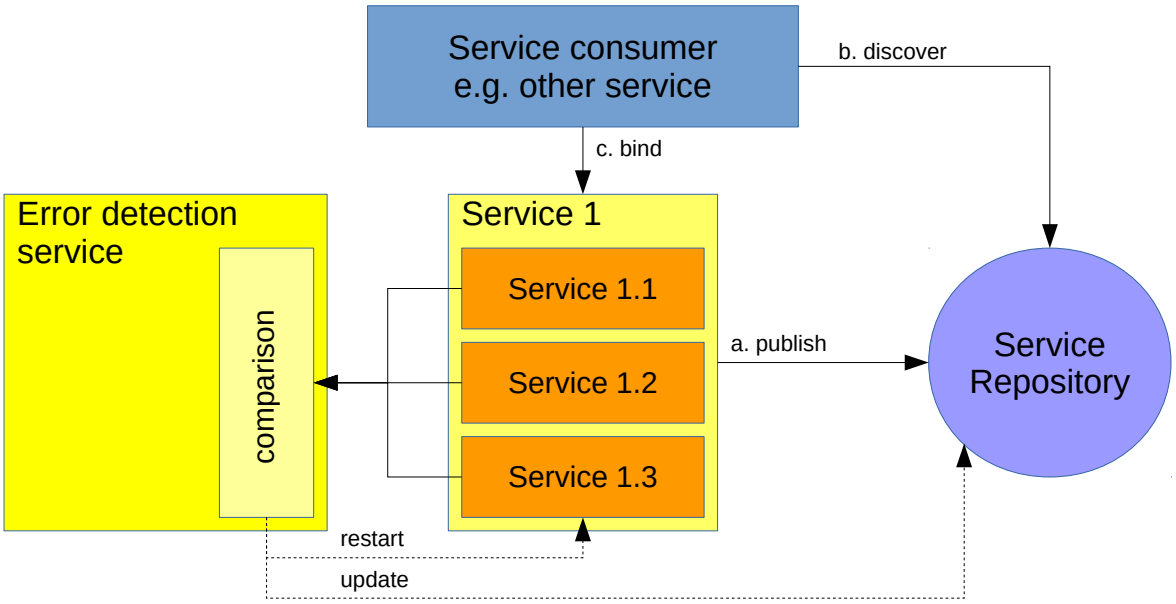


Figure 2.7: Example architecture for an *error detection service*.



Chapter 3

Results

Chapter 4

Discussion

Chapter 5

Conclusion

- still a long time until real application + many advantages of the soa paradigm - vermarkung von soa schwierig -¿ was ist der vorteil von auto mit soa + wsl unvermeidbar für selbstfahrende fahrzeuge und etc. - das problem ist safety critical ...

Bibliography

[1] Systems and software engineering - system life cycle processes, 2015.

[2] R. Omermaisser and H. Kopetz. *GENESYS - A Candidate for an ARTEMIS Cross-Domain Reference Architecture for Embedded Systems*. Vienna University of Technology, 2009.

[3] Iso 26262 road vehicles - functional safety - part 1: Vocabulary, 2011.

[4] Jerker Delsing. Deliverable d1.2 of work package 1, 2015.

[5] AUTOSAR. Glossary, autosar release 4.2.1, 2014.

[6] Iso 26262 for safety-related automotive e/e development: Introduction and overview.

[7] Wikipedia. Mpsoc. `http://en.wikipedia.org/wiki/MPSoC`. Visited: May 2015.

[8] Lund Ericsson. Emc^2 service architecture, 2015.

[9] Douglas Rodrigues, Rayner15 de Melo Pires, Júlio Cézar Estrella, Emerson Alberto Marconato, Onofre Trindade, and Kalinka Regina Lucas Jaquie Castelo Branco. Using soa in critical-embedded systems. In *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, pages 733–738. IEEE, 2011.

[10] Peter Marwedel. *Embedded system design: Embedded systems foundations of cyber-physical systems*. Springer Science & Business Media, 2010.

[11] Cesare Alippi. *Intelligence for Embedded Systems*. Springer, 2014.

[12] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[13] C. Szyperski, D. Gruntz, and S. Murer. *Component Software: Beyond Object-oriented Programming*. ACM Press Series. ACM Press, 2002.

[14] F. Kirschke-Biller. Autosar - a worldwide standard, current developments, roll-out and outlook. *15th International VDI Congress Electronic Systems for Vehicles 2011*, 2011.

[15] Serviceorientation.com. `http://www.serviceorientation.com`. Accessed: 2015-06-29.

[16] An introduction to autosar.

[17] J. Sametinger. *Software Engineering with Reusable Components*. Springer Berlin Heidelberg, 2013.

[18] Jim Q Ning. Component-based software engineering (cbse). In *Assessment of Software Tools and Technologies, 1997., Proceedings Fifth International Symposium on*, pages 34–43. IEEE, 1997.

[19] Mario Gleirscher, Andreas Vogelsang, and Steffen Fuhrmann. A model-based approach to innovation management of automotive control systems. In *Software Product Management (IWSPM), 2014 IEEE IWSPM 8th International Workshop on*, pages 1–10. IEEE, 2014.

[20] Christian Buckl, Stephan Sommer, Andreas Scholz, Alois Knoll, Alfons Kemper, Jörg Heuer, and Anton Schmitt. Services to the field: An approach for resource constrained sensor/actor networks. In *The Fourth Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE 2009) - extended version*. IEEE, 2009.

[21] Andreas Scholz, Stephan Sommer, Christian Buckl, Gerd Kainz, Alfons Kemper, Alois Knoll, Jörg Heuer, and Anton Schmitt. Towards and adaptive execution of applications in heterogeneous embedded networks. In *Software Engineering for Sensor Network Applications (SESENA 2010)*. ACM/IEEE, 2010.

[22] Stephan Sommer, Christian Buckl, Gerd Kainz, Andreas Scholz, Irina Gaponova, Alois Knoll, Alfons Kemper, Jörg Heuer, and Anton Schmitt. Service migration scenarios for embedded networks. In *The Fifth International Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE 2010)*. IEEE, 2010.

[23] Dae-Hyun Kum, Joonwoo Son, Seon-bong Lee, and Ivan Wilson. Automated testing for automotive embedded systems. In *SICE-ICASE, 2006. International Joint Conference*, pages 4414–4418. IEEE, 2006.

[24] Ashraf M El-Attar and Gamal Fahmy. An improved watchdog timer to enhance imaging system reliability in the presence of soft errors. In *Signal Processing and Information Technology, 2007 IEEE International Symposium on*, pages 1100–1104. IEEE, 2007.

[25] Shimpei Yamada, Yukikazu Nakamoto, Takuya Azumi, Hiroshi Oyama, and Hiroaki Takada. Generic memory protection mechanism for embedded system and its application to embedded component systems. In *Computer and Information Technology Workshops, 2008. CIT Workshops 2008. IEEE 8th International Conference on*, pages 557–562. IEEE, 2008.