FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Self-Healing Cyber-Physical Systems

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

Dipl.-Ing. Denise Ratasich
Registration Number 00826389

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Co-Advisor: Assoc.Prof. Dr. Ezio Bartocci

The dissertation has been reviewed by:

Axel Jantsch	Roman Obermaisser

Vienna, 18th September, 2019

Denise Ratasich

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Denise Ratasich

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 18. September 2019

Denise Ratasich

Acknowledgements

"Gratitude can transform common days into thanksgivings, turn routine jobs into joy, and change ordinary opportunities into blessings."

– William Arthur Ward

I want to thank my advisor Prof. Radu Grosu for his guidance, and for offering me a teaching assistant position which lead to a great time at the Cyber-Physical-Systems group and finally to this thesis. Many thanks go to my co-advisor Prof. Ezio Bartocci for his constructive guidance in the last two years. I am also thankful to partners from other groups, universities and industry, in particular Tiago Amorim, Dennis Kooijman, Mario Driussi, Florian Geissler, Muhammad Shafique, Edin Arnautovic, Omar Veledar, Daniel Hauer and Lukas Krammer, for their use cases, support with demonstrators, collaboration and discussions. Thanks to Bernhard and Thomas for reviewing this thesis.

Special thanks go to my friends at TU Wien. In particular, to: Bernhard, Christian and Daniel for the calming "Morgen-Meetings"; Oliver for our PhD topic, ideas and advice. Bernhard, Daniel, Ezio and Haris for inspiring discussions and for making sure that I keep up with the PhD; Gerda, Leo and Maria for making all my organizational wishes come true. Last but not least, to all my colleagues at the CPS group: You made my time at the TU Wien – magnificent!

My warmest thanks go to my family, Thomas and friends. There are no words that can express my thanks for you. If words of this thesis could be hugs I would have written much more pages. I am so thankful you are part of my life!

Kurzfassung

In unserer Umgebung werden vermehrt digitale Geräte zum Messen und Steuern von physikalischen Systemen – sogenannte *Cyber-Physical Systems* (CPSs) – eingesetzt, zum Beispiel in intelligenten Häusern, medizinischen Geräten und autonomen Fahrzeugen. Immer häufiger werden CPSs miteinander verbunden, um Geräte zu teilen und weitere Daten auszutauschen. Solche Systeme weisen eine hohe Dynamik oder Elastizität, Heterogenität und Komplexität auf, was zu einer erhöhten Verwundbarkeit und Fehleranfälligkeit beiträgt. Daher benötigen moderne CPSs adaptive Fehlertoleranz, um langfristig deren Zuverlässigkeit und Sicherheit zu garantieren.

Self-Healing ist eine relativ neue Methode, um die Widerstandsfähigkeit oder *Resilienz*, d.h. eine angemessene Überwachung und Korrektur bei Fehlern, eines elastischen Systems zu gewährleisten. In dieser Arbeit wird ein Ansatz vorgestellt, der auf Redundanz von kommunizierter Information basiert. Die Daten oder Signale von physikalischen Eigenschaften (CPS Variablen), welche in einem Netzwerk geteilt werden, können in einer Wissensbank in Relation zueinander gesetzt werden. Diese *implizite* Redundanz kann dazu genutzt werden fehlerhafte Sensoren zu erkennen und diese gegebenenfalls mit Ersatzkomponenten auszutauschen – sogenanntes *Self-Healing by Structural Adaptation* (SHSA).

Diese Arbeit entwickelt Anforderungen und Richtlinien für die Architektur eines Systems, das den SHSA-Dienst integrieren soll, und präsentiert Algorithmen, die Fehler detektieren und ausmerzen. Dazu wird eine Wissensbank abgeleitet und implementiert, welche die Relationen zwischen CPS Variablen modelliert (unter der Annahme, dass implizite Redundanz im System existiert). Die Wissensbank wird benutzt, um adaptive Monitore zu erstellen, die verwandte und ähnliche Signale vergleichen, wobei auch Ungenauigkeiten im Wert- und Zeitbereich der Signale berücksichtigt werden. Ein Monitor löst im Fehlerfall die Korrektur-Prozedur des Systems aus. Während der Korrektur wird die Wissensbank durchsucht, um eine optimale Ersatzkomponente für die fehlerhafte Information zu generieren. Weiters ermöglicht ein passendes Interface für die Ersatzkomponente die Verarbeitung von Messwerten verschiedener Quellen, welche in einem großen Netzwerk oftmals asynchron, mit unterschiedlichen Raten und verzögert empfangen werden. Die vorgeführten Anwendungsfälle und Prototypen demonstrieren die Anwendbarkeit und Funktionalität von SHSA. Eine Evaluierung zeigt, dass SHSA für viele Fehlerarten eingesetzt werden kann und die Laufzeit der Korrektur durch eine geeignete Suche im Vergleich zu verwandten Algorithmen verringert werden kann.

Abstract

An increasing number of digital devices – so-called *Cyber-Physical Systems* (CPSs) – measure and control several aspects of our physical environment. More and more CPSs are connected with each other to share devices and information, e.g., in smart buildings, smart medical devices or autonomous vehicles. However, the higher level of elasticity or dynamicity, heterogeneity and complexity adds to the system's vulnerability, thus challenges its ability to react to faults. Such a system requires adaptive fault-tolerance to ensure long-term dependability and security or scalable resilience.

Self-healing is an increasingly popular approach for ensuring resilience, that is, a proper monitoring and recovery to failures, in elastic or dynamic CPSs. This work presents a self-healing service that exploits redundancy of information on a communication network. Information or signals of physical entities – CPS variables – can be encoded in a knowledge base collecting the relations between these CPS variables. Such an *implicit* information redundancy can be used to detect and substitute failed observation components (e.g., sensors) – referred to as *Self-Healing by Structural Adaptation* (SHSA).

This work develops requirements and guidelines for the system architecture where the SHSA service shall run on and proposes algorithms that detect and mitigate failures. To this end, an adaptive knowledge base is derived and implemented by modeling relations among CPS variables given that certain implicit redundancy exists in the system. The knowledge base is then used to generate adaptive runtime monitors which compare related signals by considering uncertainties in space and time. The monitor is used to trigger the recovery process of SHSA. During recovery, the knowledge base is again used to extract an optimal substitute – optimal, w.r.t. a user-defined utility function – through guided search considering properties of signals, variables and relations. Moreover, a proper interface of the substitute enables the processing of asynchronous, multi-rate and delayed measurements of different sources. The presented use case discussions and real-world prototypes feature the applicability and functionality of SHSA. The evaluation shows the ability of detecting various kinds of faults and an increased runtime performance of the substitution search compared to related work.

Contents

CHAPTER 1

Introduction

1.1 Motivation

As digital devices get ubiquitous in our daily life and more and more connected with each other, our society comes up with tons of applications to exploit the information generated and communicated within a network. A growing number of applications use systems that are tightly connected with the physical world, so-called Cyber-Physical Systems (CPSs) [34].

The behavior or service of a CPS – may it be implemented as a simple or sophisticated controller featuring Artificial Intelligence (AI) – depends on its inputs. The inputs to a CPS are typically sensor measurements or derivations of it, and suffer from uncertainty (e.g., noise or the unpredictability of the environment) and the possibility of faults (e.g., unreliable hardware, security breaches or faults triggered by the evolution of the system) [50, 3]. When the inputs are faulty, the controller may not provide its intended service or even lead to fatal consequences. Recent prominent examples are the crashes of two Boeing 737 Max [19] where the automated system MCAS – a controller to stabilize the airplane to avoid a stall – failed due to faulty inputs. For such a safety-critical service, Boeing used only *a single* sensor as input, measuring the angle of attack by a mechanical wing, mounted at the outside of the plane and therefore exposed to likely environmental hazards (like icy weather and birds).

Consequently, to avoid malfunction, i.e., failures, and fatal consequences to humans or the environment, systems – and especially CPSs – require to be resilient [3, 6, 60, 14]. A *resilient* system persists its service delivery that can be justifiably trusted, even when facing changes [33].

The backbone of modern CPSs is the so-called Internet of Things (IoT) [2, 57, 61, 60]. This ultra-large-scale network connects billions of devices[1] providing an uncountable

[1] https://www.statista.com/study/27915/internet-of-things-iot-statista-dossier/ (IHS, ID 471264)

number of bytes of data every day. It provides means to conveniently access devices and extract or exchange information, e.g., for monitoring, configuration, control and maintenance. The IoT is exploited in various intelligent or *smart* CPS applications of different domains [2, 57, 49, 60], e.g., smart mobility, smart manufacturing, smart agriculture, smart buildings or smart healthcare. Several CPSs may be connected to and use the IoT simultaneously. A CPS might also share *things* (e.g., sensors) with other CPSs. This system-of-systems will also undergo changes over time, especially when subjected to long operational duration (over decades like in autonomous vehicles). It therefore should scale dependability and security when it comes to functional, environmental and technological changes [32] – referred to as long-term dependability and security.

To cope with the unpredictability and evolvability of such systems while ensuring its service, the system itself shall be *adaptive* [28, 12]. Traditional fault-tolerance is typically limited to the faults and threats considered during design time [18]. For instance, simple fault-tolerance may be voting over a result of redundant components or resetting a task on error. Nevertheless, to remain safe and secure even in the advent of faults and threats that have not been considered or predicted at design time but could emerge during runtime, the system needs means to *heal itself*.

Self-healing [47, 18] is a promising approach to cope with unforeseen faults which requires the notion of self-awareness [35] and self-adaptation [12]. Self-aware computing systems learn the models of the system itself and its environment to reason and act (adapt) – here: self-healing (detect and recover) – in accordance to higher-level goals (here: resilience)[31]. The key feature of self-* or self-X techniques is continuous learning and optimization which is performed during runtime to evolve the models upon system changes.

1.2 Problem Statement and Research Gap

Self-healing evolved from traditional fault-tolerance by adding autonomy and AI. The process is therefore more complex than traditional fault-tolerance and can be split into several parts. For instance, MAPE-K is a common framework for self-adaptation, originally developed to automate (any) tasks [28]. It expresses the feedback loop – Monitor, Analyze, Plan and Execute using some Knowledge – of a self-adapting system. Related work may rename stages (to, e.g., collect, analyze, plan and act in self-adaptation roadmaps [12, 13]) or summarize stages (to, e.g., monitor, diagnose and recover in self-healing surveys [47]). These stages are typically studied in different research areas (e.g., detection and recovery). The term self-healing has been coined just recently by the software community working in the area of autonomic computing [28]. Though, several roadmaps [12, 13, 59] review the evolution of self-adaptation and elaborate on the future research directions in self-adaptation, only a very view surveys exist focusing on self-healing in software systems [18, 47]. Especially in the area of CPS and IoT the topic of self-healing (and self-adaptation) is rarely discussed.

Self-adaptation or self-healing often requires a fair amount of flexibility w.r.t. the architecture. However, this often contradicts with the setup used in CPS applications.

Many CPSs must adhere to deadlines, i.e., implement real-time applications [30]. CPS architectures and platforms (like microcontrollers or time-triggered networks) are therefore often configured statically (e.g., programs and communication channels do not change; the control task is executed periodically). Replacing the communication backbone of a CPS by the large-scale and elastic IoT, raises several challenges regarding the architecture and interface. For instance, the interface of a CPS application has to cope with asynchronous, multi-rate and/or delayed messages of an elastic communication network. A pressing need is therefore to define guidelines and implement architectures to ease up self-healing for CPS.

"Self-adaptation" raises red flags in every engineer developing safety-critical CPS. A fault in a safety-critical system can lead to fatal and even life-threatening consequences. To avoid such situations, risks of faults in the system are analyzed, i.e., severity and likelihood determined, and appropriate fault-tolerance mechanisms are designed and implemented [30, 24]. Finally the whole system is verified and certified. Changing the system during runtime, however, would disable verification and certification. Moreover, due to the complexity and elasticity of the systems not every critical situation or possible fault can be foreseen. Traditional design-time methods for fault-tolerance and verification of a CPS are therefore increasingly difficult to apply. Self-healing, that is the adaptation to faults during runtime, is an opportunity to overcome unforeseen faults. In addition to self-healing, CPSs will require proper runtime verification [4, 51].

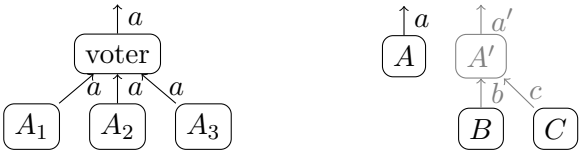
1.3 Aim of the Thesis

This thesis develops solutions for self-healing CPSs. From a methodological point of view, the following question comes first.

RQ 1: *What state-of-the-art self-healing is relevant for CPSs? What are the challenges and possibilities of self-healing in a CPS that is part of the IoT?*

Most of the state-of-the-art literature on self-healing [18, 47] can be split into surveys on anomaly detection [11, 37, 9, 8], runtime monitoring [4, 17], fault-tolerance [3, 30, 24, 21], dynamic reconfiguration [58] and self-adaptation [12, 13, 59, 31]. Questioning the applicability of these methods to CPSs might narrow the pool of potential approaches. Recall, some properties of CPSs, notably real-time performance and safety-criticality, require special treatment and caution. Yet, the connectivity enabled by the IoT can also raise new possibilities (e.g., more things, more information).

A very promising way is Self-Healing by Structural Adaptation (SHSA). SHSA replaces a failed service with a *substitute* by exploiting implicit redundancy. In contrast to explicit redundancy which is achieved by replicating critical system services and voting over the outputs (Fig. 1.1a), some failed information may be derived from related information too (Fig. 1.1b). For instance, the information a, provided by service A, can be substituted by a', which is derived from the combination of b and c.



(a) explicit (b) implicit

Figure 1.1: Types of redundancy.

A designed system might not incorporate (implicit) redundancy straightaway (if so, the system would use information fusion to exploit the redundancy), however, when assembling (sub-)systems of different suppliers redundancy often becomes available. This is typically the case for a distributed system or system-of-systems, e.g., a (connected) vehicle or in general CPSs connected via the IoT.

The foundations of this work are laid by Höftberger [22] who developed a knowledge base (in the form of an ontology) describing implicit redundancy in a system. This work applies and enhances Höftberger's work on self-healing exploiting redundancy by discussing the following research questions.

RQ 2: *How to enable and integrate SHSA in CPSs?*

SHSA is a system-wide approach, i.e., it is not limited to a single component. It uses available information exchanged through the communication network of a system to generate a substitute for a failed service during runtime. Hence, the capabilities of the network and the adaptability of the system play a crucial role.

RQ 2.1: *What are the architectural requirements for SHSA?*

Keeping the architectural requirements in mind, a designer can build the system accordingly or determine if SHSA is feasible for an existing system. In particular, the relevant platform components e.g., hardware, operating system, network, etc., and its needs to enable SHSA shall be identified.

For instance, the IoT is pushed as network for CPSs, though its architectural and communication paradigms mostly differ. CPSs are typically configured during design time. That is the tasks or processes run on a specific controller or CPU and (periodically) consume from and provide information to dedicated devices (respectively sensors and actuators). The IoT, on the other side, implements a loosely-coupled, elastic network of things – similar to the Internet where services can connect and disconnect during runtime. The paradigms of both worlds may be more or less suitable for SHSA.

RQ 2.2: *How to handle asynchronous, multi-rate, late or even missing messages?*

CPS components often assume the observations to be taken at a common point in time, that is *synchronously*. For instance, sensor fusion and state estimation components periodically sample and combine measurements to improve the believed state of the physical system [36, 56, 52]. Substitutes presented in this work as well as traditional

monitors and voters (e.g., Triple Modular Redundancy (TMR) [30]) rely on synchronous observations too. Yet, large and distributed sensor networks or the IoT often use asynchronous communication paradigms (i.e., measurements are communicated via messages that are not synchronized in time) and may suffer from latency, jitter or even package loss. Moreover, different sensors provide their measurements with different rates (cf. gyroscope and camera).

RQ 3: *How to detect faults or failures, and trigger SHSA?*

SHSA requires a trigger by a monitor or fault detection unit. Failures are typically detected by specifying the desired behavior in the form of a model [11, 37, 9], signature [8] or specification [4].

A simple method, linked to SHSA, for detecting a faulty sensor value in different failure scenarios is to check against related information sources [30, 46, 44, 15, 38]. Unfortunately, the information communicated in the system (particularly from diverse components) suffers from uncertainties in space (e.g., noise) and time (e.g., delays). Subscribed or received information therefore cannot be directly compared.

RQ 4: *How to speed up the recovery?*

The dynamics of a CPS typically require sample and processing rates of ≤ 100 ms (e.g., motors, autonomous vehicles). It is therefore desired to mitigate a failure within such time frame.

Höftberger [22] applied simple Depth-First Search (DFS) to find related information in the knowledge base in case some information fails. The first possible relation found is used to generate the substitute. However, the vast amount of available information in the IoT may slow down the substitution search. Furthermore, the setup of the substitute or shutdown of an existing service can take a significant amount of time. It is therefore desired to quickly find and start the *best* or *optimal* substitute right away.

RQ 5: *How to evaluate and select a substitution?*

In case of more than one possible substitutions, SHSA needs means to evaluate and choose the optimal solution for the application. In addition, SHSA has to take care that the solution satisfies the application's requirements, e.g., regarding safety.

1.4 Methodology

The last section presents this work's goals in the form of research questions serving as a starting point of the methodological approach this thesis follows. The research questions have been refined to search, choose from, define and implement *i)* models for the knowledge base and the selection of substitutions, *ii)* methods to search for and evaluate substitutions and *iii)* architectures to implement and run SHSA. The models and implementations are finally evaluated by appropriate test data and comprehensive use cases.

Requirements and State-of-the-Art. As the term self-healing is relatively new, the terminology has been clarified first by investigating papers about dependability and resilience [3, 32, 33], subsequently about fault-tolerance [3, 30, 24], self-healing [18, 47] and self-adaptation [12, 13, 52, 59, 31] in CPS and software engineering to acquire search strings and related work to this topic.

The lesson learned during the continuous search for related work is that self-healing is a broad field including detection, diagnosis and recovery [47], each likely to use different models, tools and approaches [11, 24, 59, 4] (Ch. 2). For instance, fault detection can be achieved by specifying the allowed signature [8], modeling the anomaly [11] or checking against redundancy [30, 15, 38]. Moreover, the approach differs w.r.t. goal (dependability vs. security), failure model (noise vs. strategic attack) or domain (CPS vs. cloud services).

Building on top of the work from Höftberger [22] the focus of further state-of-the-art research therefore was on models and methods for redundancy-based SHSA in CPS. Because the models and methods for self-healing typically originate from AI [52] or Autonomic Computing (AC) [28] the requirements of a CPS have been kept in mind throughout the research and development presented in this work. For instance, a CPS might be resource-constrained and implement real-time and safety-critical applications.

One goal of this work was to apply and integrate SHSA into CPSs and show the applicability of SHSA in a CPS. This required a suitable platform and at least one comprehensive use case.

Architecture and Platform. To build a suitable platform for SHSA in CPS, the requirements of SHSA w.r.t. the architecture have been clearly defined (Ch. 3). A search for platforms and applicable frameworks [53, 2] including process and communication management (middleware, protocol, etc.) based on these requirements has been conducted. Another important aspect in the selection of a SHSA-enabling framework was the use case which has been chosen to be a mobile robot. The prototypes therefore used the Robot Operating System (ROS) [48], a widely used framework in robotics which fulfills all SHSA requirements and provides nice-to-have features for prototyping and further development (e.g., tools to debug and visualize the application, interfaces to other frameworks).

The architecture of the mechanism has been selected based on related work in self-adaptation [59], AC [31], AI [52] or CPS [34, 30]. For instance, the process and the components of the self-healing mechanism have been developed according to Monitor, Analyze, Plan and Execute using a Knowledge base (MAPE-K) [28], a well-established framework for automating tasks and widely used in the AC or self-* community [31].

Models. A starting point for the model of redundancy was the knowledge base defined by Höftberger [22]. The knowledge base models the relations between CPS variables. As CPS variables are random variables, probabilistic models have been considered, e.g.,

Bayesian or Markov networks [52, 29]. However, the substitution follows a rule-based approach determined by the relations between variables (Ch. 5).

To be able to pick the right substitution, the substitution and its performance measures have been formally specified. Models and methods have been investigated towards their suitability to extend the knowledge base, also considering different communities, e.g., verification [4] (e.g., contract-based approaches [5, 20]), decision making and artificial intelligence [52] (e.g., utility theory).

Methods. The same procedure has been applied for the methods. Approaches using rules, graphs, probabilistic models, AI and verification have been researched, filtered and combined to apply self-healing in CPS. For instance, a graph-based approach (well established and efficient) is applied to search substitutions, utility theory (originates from decision making in AI) is exploited to select a substitution (Ch. 6) and contracts (typically used in verification) are used to verify a substitution (Ch. 7).

Evaluation. Testing and evaluating self-healing in CPS requires data from physical entities i.e., datasets (e.g., acquired in a sensor network), simulators or prototypes providing samples over time. Hence, an online search for appropriate simulators or benchmarks and datasets on platforms like UCI Machine Learning Repository [10] or Kaggle [26] has been conducted. Unfortunately, the available datasets on different platforms lack data from sensors measuring different physical entities. The available datasets often represent discrete data (e.g., for classification), extract features from a single sensor source (e.g., image sensors) or collect data of replica sensors (e.g., temperature at different sites). To the best of my knowledge, there are also no (open or free) simulators providing *related* signals that could be used for evaluation. Moreover, a major part of this work includes the design of the architecture and implementation of redundancy-based self-healing of observation capabilities of CPSs which is best evaluated on a real-world CPS prototype.

Therefore, this thesis evaluates the developed solutions on a real-world CPS prototype of a mobile robot (instead of investing resources into the implementation of a simulator). The robot comprises various sensors and computing units forming a heterogeneous system-of-systems similar to the IoT. Several other use cases with research and industrial partners are discussed in Chapter 2 and 7. In addition, where applicable and/or necessary, artificial data has been generated, e.g., to check the complexity in space and time of the implementations (Ch. 3 and 6).

Throughout the implementation of SHSA and the prototypes, state-of-the-art development methods and tools have been used to keep the research reproducible (documentation, version management, unit testing, scripts, online code and experiments, virtual environments and Docker images for testing and deployment, etc.).

Issues and Limitations. The presented example knowledge bases in the use cases might seem quite small. On the one side this provides a fast substitution result, on the

other side SHSA may seem to be an additional overhead. Yet, SHSA provides a way to model and exploit redundancy which can be extended, adapted and reused. Also note that the presented knowledge bases focus on the application and are limited by the prototypes (cf. the number of sensors on the mobile robot with an electric vehicle).

This work provides requirements, guidelines, formal definitions and algorithms of SHSA. Note that the presented platform and implementations are prototypes only. The selection of the platform for SHSA or parts of its implementations might have to be reconsidered with new architectures and frameworks coming up, e.g., the selection of ROS as the platform for SHSA or the processing of asynchronous, multi-rate or delayed measurements, respectively. Meanwhile a second version of ROS is available (ROS 2.0 [43]) and the industry is also working on middlewares and standards towards dependable message transfer in CPS (e.g., Industrial IoT [60] or Time-Synchronized Network (TSN) in the Ethernet standard [27]). Nevertheless, in IoT and wireless sensor networks the work on temporal alignment is still of utmost relevance.

SHSA has been developed to mitigate failures of observation services. Failures of controllers and actuators are not handled in this work. Failures manifest at the output of services. Given a proper description or semantics of the controllers' outputs, the SHSA knowledge base may also handle redundancy of controllers. The majority of CPSs implement a single controller specific to their application (an exception is the domain of avionics where TMR or explicit redundancy is applied). In the IoT which incorporates several CPS applications, each equipped with a separate controller, modeling the redundancy of controller outputs can enable resilience to these components too. This requires further research and standardization on the semantic description of service interfaces. Possible starting points are IoT ontologies, e.g., SensorML [7]. The output of an actuator controls a physical entity and is not an information transmitted in the communication network any more. Because SHSA substitutes failed information on the network, actuator failures cannot be mitigated by SHSA.

1.5 Synopsis

This work is written as a cumulative dissertation that is a collection of published articles. Chapters 2 to 7 represent following publications:

Chapter 2 (Roadmap) Denise Ratasich, Faiq Khalid, Florian Geissler, Radu Grosu, Muhammad Shafique and Ezio Bartocci. A Roadmap Toward the Resilient Internet of Things for Cyber-Physical Systems. *IEEE Access*, 7(1):13260–13283, Jan. 2019.

Chapter 3 (Architecture) Denise Ratasich, Oliver Höftberger, Haris Isakovic, Muhammad Shafique and Radu Grosu. A Self-Healing Framework for Building Resilient Cyber-Physical Systems. *In 2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 133–140, May 2017.

Chapter 4 (Fusion) Denise Ratasich, Bernhard Frömel, Oliver Höftberger and Radu Grosu. Generic sensor fusion package for ROS. *In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 286–291, Sept. 2015.

Chapter 5 (Detection) Denise Ratasich, Michael Platzer, Radu Grosu and Ezio Bartocci. Adaptive Fault Detection Exploiting Redundancy with Uncertainties in Space and Time. *In 2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 23–32, June 2019.

Chapter 6 (Recovery) Denise Ratasich, Thomas Preindl, Konstantin Selyunin and Radu Grosu. Self-healing by property-guided structural adaptation. *In 2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pages 199–205, May 2018.

Chapter 7 (Assurance) Tiago Amorim, Denise Ratasich, Georg Macher, Alejandra Ruiz, Daniel Schneider, Mario Driussi and Radu Grosu. Runtime Safety Assurance for Adaptive Cyber-Physical Systems: ConSerts M and Ontology-Based Runtime Reconfiguration Applied to an Automotive Case Study. *Solutions for Cyber-Physical Systems Ubiquity*, pages 137–168, 2018.

Section 1.6 summarizes the listed publications. This introduction (including the summary) references the chapters – not the publications – to distinguish the author's work to related work and further references.

Note, the chapters of this electronic version of the thesis contains the abstract, citation and a link to the papers only.

1.5.1 Research Questions Covered per Chapter

The Chapters 2 to 7 develop solutions to the research questions given in Section 1.3 (Table 1.1).

Chapter 2 (Roadmap) introduces the terminology to faults and resilience, summarizes the state-of-the-art regarding self-healing (in particular detection, diagnosis and recovery) of failures, outlines the challenges of long-term dependability and security and a roadmap to achieve resilience over the life-cycle of a CPS. Chapter 3 (Architecture) presents the architectural requirements of SHSA, lists design guidelines and presents an implementation on top of the ROS in an use case of a mobile robot. Chapter 4 (Fusion) deals with the implementation of an interface for sensor fusion in ROS. It showcases how a service requesting synchronous messages can deal with asynchronous, multi-rate and delayed messages. Chapter 5 (Detection) implements the knowledge base as a set of rules in Prolog and proposes an observation model and algorithm to compare signals against each other for fault detection. Chapter 6 (Recovery) describes a formal model of the knowledge base of SHSA, an extension to evaluate possible substitutions and a guided-search algorithm to find the best recovery strategy. Chapter 7 (Assurance) deals with the verification of self-adaptation using safety certificates that are contracts between application or platform

	Research Question	Publication (Ch.)
RQ 1	What state-of-the-art self-healing is relevant for CPSs? What are the challenges and possibilities of self-healing in a CPS that is part of the IoT?	Roadmap (2)
RQ 2	How to enable and integrate SHSA in CPSs?	Architecture (3)
RQ 2.1	What are the architectural requirements for SHSA?	Architecture (3)
RQ 2.2	How to handle asynchronous, multi-rate, late or even missing messages?	Fusion (4)
RQ 3	How to detect faults or failures, and trigger SHSA?	Detection (5)
RQ 4	How to speed up the recovery?	Recovery (6)
RQ 5	How to evaluate and select a substitution?	Recovery (6), Assurance (7)

Table 1.1: Research questions covered per chapter.

components to be satisfied. The fulfillment of the contracts is checked during runtime, in particular, when the system is reconfigured, here by SHSA. An automotive use case demonstrates the approach of SHSA safety assurance.

1.5.2 Chapters Mapped to the MAPE-K Framework

The different chapters tackle various aspects of self-healing and specifically SHSA, which can be roughly divided into the architecture or interface for self-healing, fault detection and recovery. Figure 1.2 maps the articles to the MAPE-K framework (recall: monitor/detect, analyze/diagnose, plan and execute/recover) which represents the feedback loop of automating a task, here self-healing.

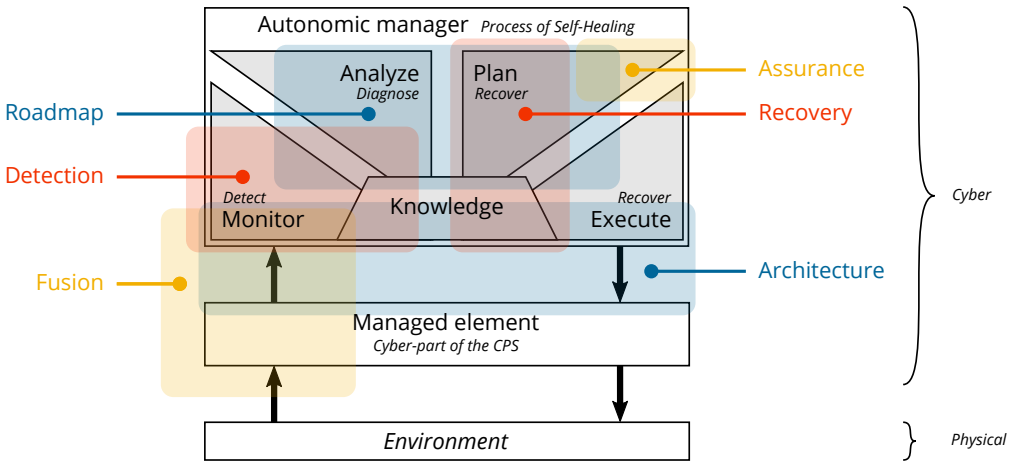


Figure 1.2: Articles mapped to the MAPE-K framework. Terms used in self-healing and CPSs are written italic.

The "Roadmap" article gives an overview of available detection, diagnosis and recovery approaches. The articles "Architecture" and "Fusion" particularly look into the required architecture and the interfaces of self-healing services (e.g., monitors or substitutes) to other system services. "Detection" develops a redundancy-based monitor and re-implements the SHSA knowledge base. "Recovery" extends the knowledge base and improves the substitution search for a failed service. "Assurance" showcases the verification of the planned recovery given some safety contracts between services.

1.5.3 Attributes of the Publications

All publications have been peer-reviewed (Table 1.2).

Publication	Type	Rank	PR	Author	Contribution
Roadmap	in journal IEEE Access 2019	36/3.5	yes	1st	main
Architecture	in proceedings ISORC 2017	B1/55	yes	1st	main
Fusion	in proceedings IROS 2015	A1/45	yes	1st	main
Detection	in proceedings SASO 2019	B3/30	yes	1st	main
Recovery	in proceedings ICPS 2018	n/a/65	yes	1st	main
Assurance	in book IGI Global 2018	n/a	yes	2nd	significant

Table 1.2: Meta-data to the publications included in this work (PR .. peer-reviewed).

Table 1.2 includes common rankings of the journal or conference of the published papers, retrieved in March 2019. The ranks of the conferences are according to the Qualis conference ranking from `http://www.conferenceranks.com/`. The proceedings of the conferences provide the acceptance rates (in percent). The h-index and impact factor of the journal is extracted from `https://www.scimagojr.com/journalrank.php` and the journal's website, respectively.

1.5.4 Contribution of the Author per Chapter

Chapter 2 (Roadmap) The author's contribution is the collection of the terminology on faults and resilience, state-of-the-art of self-healing (in particular the recovery and mitigation) and its taxonomy, challenges and a roadmap for self-healing in IoT for CPS and the SHSA approach applied on a smart vehicular network use case. The work is extended with state-of-the-art of runtime verification and monitoring by Ezio Bartocci, and security-related threats and methods by Faiq Khalid and Muhammad Shafique. Faiq Khalid and Muhammad Shafique provided an initial version on the challenges of long-term dependability and security which has been refined by Ezio Bartocci and the author of this thesis. The case study has been developed in close collaboration with Florian Geissler (Intel) who provided the Matlab model and data for the SHSA approach. Faiq Khalid added security-related anomaly detection to the roadmap and case study.

Chapter 3 (Architecture) The author's contribution is the integration and application of Höftberger's Ontology-based Runtime Reconfiguration (ORR) [22] under guidance of Oliver Höftberger, and architectural requirements and design guidelines of SHSA showcased on a mobile robot. An initial version of this work has been refined by discussions with Haris Isakovic and under guidance of Muhammad Shafique and Radu Grosu.

Chapter 4 (Fusion) This work is based on the master thesis of the author of this work supervised by Oliver Höftberger and Radu Grosu. An initial version of this work has been refined by discussions with Bernhard Frömel and under guidance of Oliver Höftberger and Radu Grosu.

Chapter 5 (Detection) The author's contribution is the development and implementation of the observation model and fault detection, resulting from several discussions with Ezio Bartocci. Michael Platzer contributed with implementations to the prototype. An initial version of this work has been refined under guidance of Ezio Bartocci and Radu Grosu.

Chapter 6 (Recovery) The author's contribution is the formalization of the knowledge base for SHSA and the implementation and evaluation of a guided-substitute-search. The model has been developed in close collaboration with Thomas Preindl and Konstantin Selyunin. An initial version of this work has been refined under guidance of Radu Grosu.

Chapter 7 (Assurance) The author's contribution is the part on runtime reconfiguration or self-adaptation (mainly in the background, state-of-the-art and use case section). The case study presented in this work is the outcome of a demonstrator developed in close collaboration with the partners of the European project EMC^2, however, this work mainly describes the research of Fraunhofer (safety certificates) and TU Wien (adaptation).

1.6 Summary of the Thesis

This work discusses self-healing in Cyber-Physical Systems (CPSs) and presents Self-Healing by Structural Adaptation (SHSA).

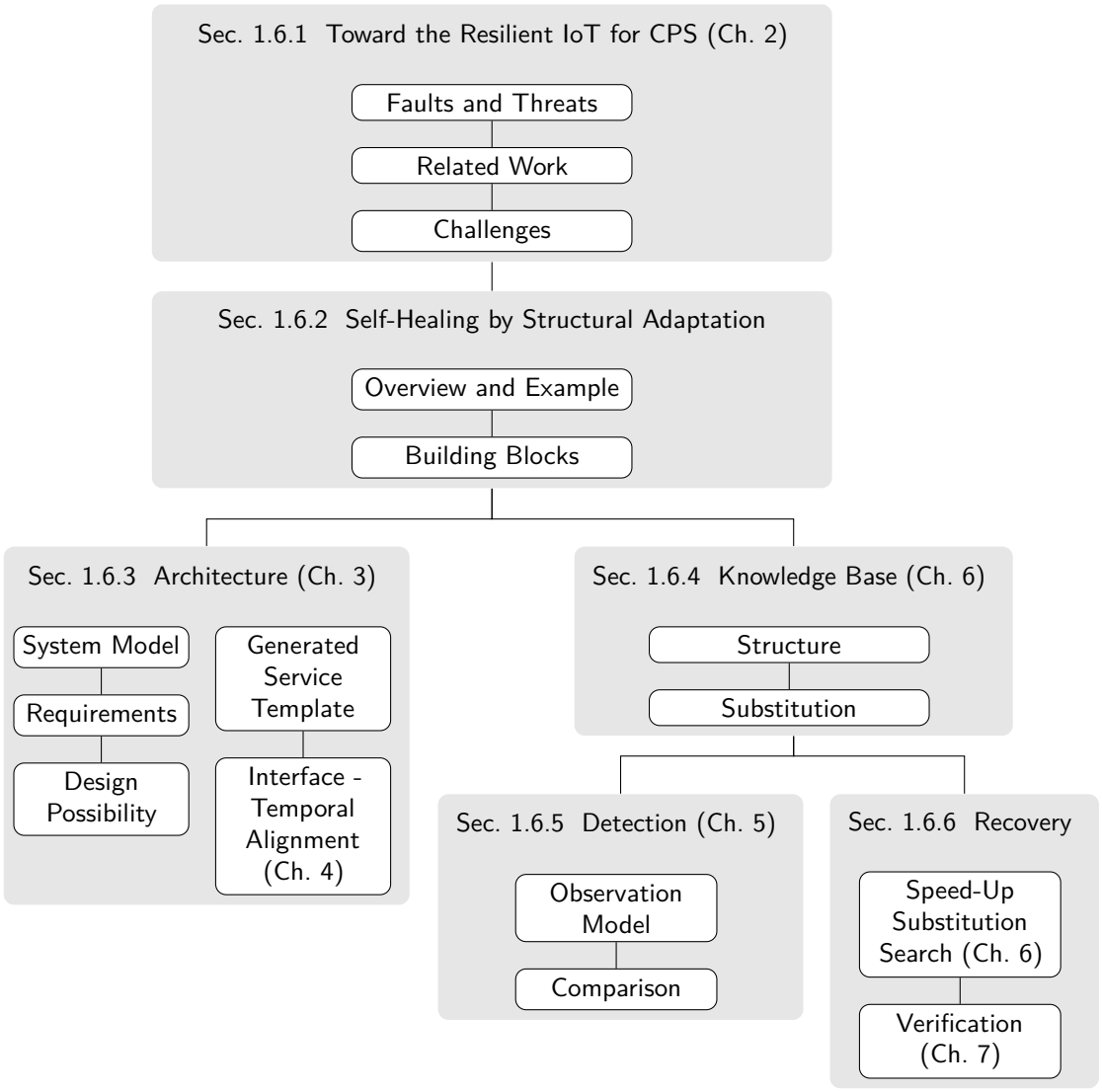


Figure 1.3: Sections of this summary and references to the chapters for details.

Figure 1.3 depicts an overview of the summary. Section 1.6.1 lists typical faults and state-of-the-art methods to detect, diagnose and recover from failures and its challenges in Internet of Things (IoT) and CPSs. Section 1.6.2 introduces SHSA. Section 1.6.3 provides the requirements and a design possibility for an architecture that enables structural adaptation. Moreover, it presents a template for services to start at runtime (e.g.,

SHSA substitutes) and its interface which enables temporal alignment of asynchronous and multi-rate inputs. Section 1.6.4 states the knowledge base modeling information redundancy. Section 1.6.5 presents an observation model to detect failures in information or signals exchanged between components and an algorithm to compare related signals to detect faults. Section 1.6.6 develops a greedy algorithm which speeds up the substitution search, and showcases a contract-based method to verify a planned substitution.

1.6.1 Toward the Resilient IoT for CPS

The devices connected by the IoT are typically part of several CPSs. In contrast to traditional embedded systems, such CPSs are heterogeneous and complex, and also dynamic in its structure and behavior (components connect, upgrade and disconnect during runtime). Such dynamicity or elasticity adds to the system's vulnerability (see long-term fault examples in Fig. 1.4). Moreover, as the IoT enables remote monitoring and control, the concepts of dependability and security get deeply intertwined. The goal of a self-healing service is therefore to provide long-term dependability and security.

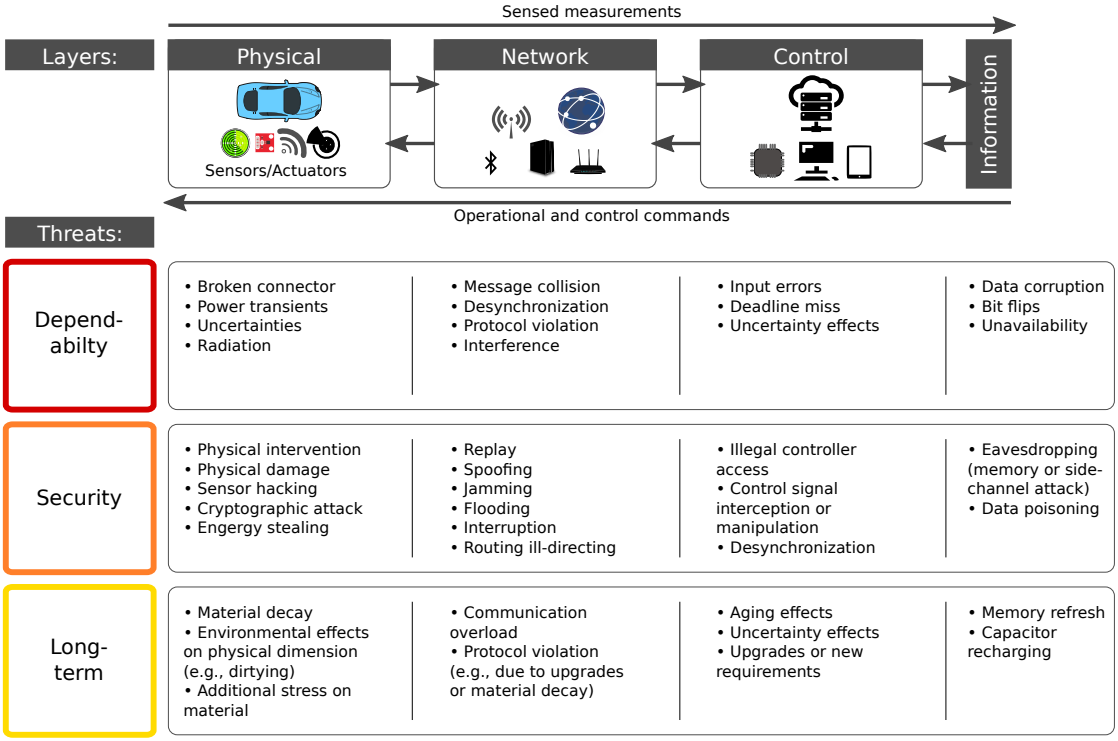


Figure 1.4: Example faults and threats with respect to CPS layers (Ch. 2).

Traditionally, dependability approaches can be distinguished from fault prevention, tolerance, removal and forecasting [3], which ensure the system's functionality in the presence of faults or mitigate failures by an appropriate design. The mechanisms may be applied during runtime or maintenance. This work focuses on adaptive mechanisms

applied during runtime to detect and recover from failures in information communicated, e.g., through the IoT. Adaptation of the mechanism itself is necessary to tackle the evolution of the system and to be able to also handle failures that were unknown during design time or could emerge during runtime.

In self-healing, typically, a knowledge base (e.g., a learned model) is used to detect and diagnose faults and recover from failures. Naturally, existing methods to fault detection and recovery are extended with self-awareness and self-adaptation capabilities to properly update the model and adapt during runtime on changes.

Overview of Existing Work that enables Resilience

The abnormal behavior or service failure can be detected by comparing the actual behavior with the behavior of redundant services, a specification or model of the desired behavior, or a signature of the anomalous behavior (Fig. 1.5).

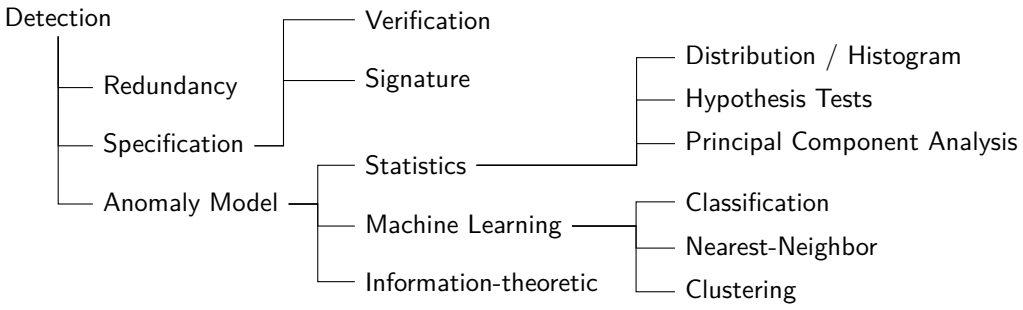


Figure 1.5: A taxonomy of methods for fault detection (Ch. 2).

The recovery from failures can be performed by adapting the parameters of a system or using redundancy in space or time (Fig. 1.6). Another possibility is runtime enforcement which enforces a program to run according to a specification, or adapts an input to conform to a specification in the form of, e.g., a finite state machine or automaton [16].

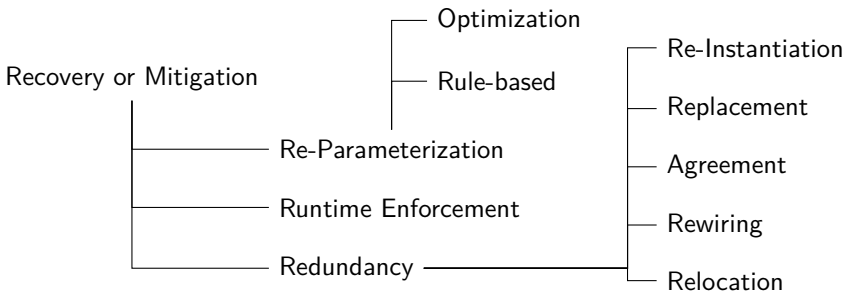


Figure 1.6: A taxonomy of methods for recovery or mitigation (Ch. 2).

Challenges

From a technological point of view some of the existing resilience methods are not directly suitable for the IoT or a CPS. Some of the major technological challenges for resilience are (Ch. 2):

- **Resource Limitations.** Most of the IoT devices are resource constrained (e.g., sensors), i.e., its processing capabilities and energy budget are optimized for their specific purpose (e.g., acquisition, pre-processing, encryption and communication of a measurement). Such devices may not allow to run sophisticated self-healing algorithms.
- **Heterogeneity and Complexity.** The heterogeneity and complexity of the system challenges not only the interoperability of the devices itself but also the resilience mechanisms which have to heal devices with different interfaces, various types of information and unspecified behavior.
- **Heterogeneity and Sharing.** Devices of an IoT may be shared between CPS applications possibly requesting different Quality-of-Service (QoS) of the devices regarding dependability and security, e.g., sample rate, accuracy, level of encryption. The resilience methods therefore must also consider the needs of different applications and the value of trust of the information when monitoring services or recovering from failures.
- **Real-Time and Scalability.** The resilience mechanism has to cope with the dynamicity, growth and evolution of the system while ensuring time constraints of CPS applications (e.g., deadlines) when probing the information or adapting the system.

The complexity, elasticity and evolution of the network might create new faults (cf. zero-day malware). Ensuring *long-term* dependability and security raises following additional challenges:

- **Self-healing.** Detect and mitigate unforeseen failures, and avoid fault propagation.
- **Guarantee resilience.** When using Artificial Intelligence (AI) or Machine Learning (ML).
- **Sustainability.** Ensure resilience over time in a resource-efficient manner. This includes minimizing the vulnerabilities and building robust resilience mechanisms.

An essential part to address long-term dependability and security (handle unforeseen faults) is learning and optimization. The mechanisms therefore require verification and validation to ensure the integrity of the system, and a robust design to cope with the elasticity of future systems (Ch. 2).

1.6.2 Self-Healing by Structural Adaptation

CPSs are typically assembled from several *subsystems* connected by a communication network (e.g., IoT) which is a subsystem on its own. Each subsystem incorporates various hardware and/or software components. A *component* is a subsystem considered as a black box with well-defined interfaces. Each component provides services that may rely upon services of other components. A *service* is the intended behavior of a system/component. Typically, services interact with each other through messages.

CPS services implement a feedback loop of observing and controlling its environment. As a consequence, the messages communicated within a CPS typically contain information calculated by controllers or collected by sensors measuring physical entities – the CPS variables. In particular, observation services publish (typically in a periodic fashion) the value and timestamp of a CPS variable v – an *observation* or snapshot of a variable at a certain point in time – via the communication network. A *signal* $v(t)$ or short v represents the values of a CPS variable v over time.

Especially in the IoT for CPSs, when assembling (sub-)systems of different suppliers, explicit and implicit redundancy of CPS variables often becomes available. Figure 1.7 depicts such information redundancy. For instance, two services output or provide signals v_{a_1} and v_{a_2} of the same kind (explicit redundancy). The implicit form of redundancy relates *different* kind of information to each other using a function. Such information redundancy can be used to compare related observations or to substitute failed observation services (e.g., sensors or state estimators), referred to as Self-Healing by Structural Adaptation (SHSA).

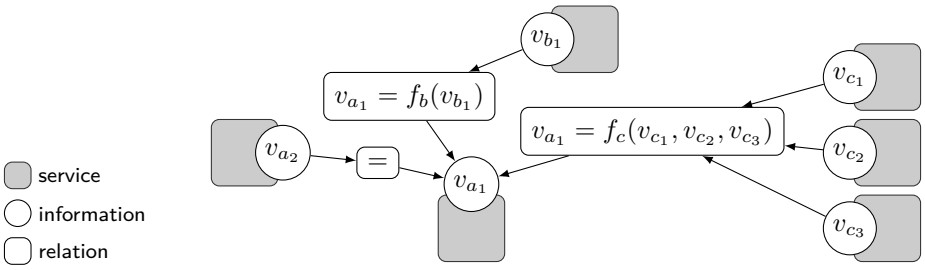


Figure 1.7: Services providing related information through its interfaces (relations can be arbitrary functions, learned or defined by a domain expert).

SHSA, in general, is a service ensuring the resilience of the system by adding or removing services or changing the information flow between services.

Example. To give an idea of how SHSA works, consider the mobile robot in Figure 1.8 equipped with several sensors to avoid obstacles (emergency brake assistant), park or recognize surrounding objects (lane-change assistant).

Like in a modern car, some features or applications (e.g., lane-change assistant) are available or not. The three applications of the mobile robot are implemented by several

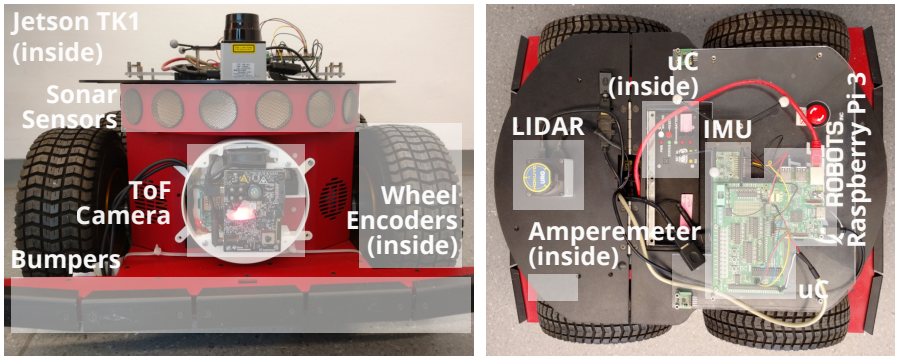


Figure 1.8: Mobile robot equipped with its sensors and processing units – the CPS prototype used in several chapters which models an autonomous car.

services. Figure 1.9 depicts the services and information flow. The robot stops in front of obstacles when the minimum of the Light Detection and Ranging (LIDAR)'s distance measurements falls below a certain threshold, parks using an array of sonar sensors and recognizes objects with a Time-of-Flight (ToF) camera. In particular, the LIDAR provides the distance measurements v_{lidar}. The Obstacle Estimator calculates the minimum distance v_{dmin} from the LIDAR. The Collision Avoidance decides if its safe to go on or if the robot should be stopped. It provides the desired speed and direction command v_{safe_cmd} to the service Robot which controls the wheel motors.

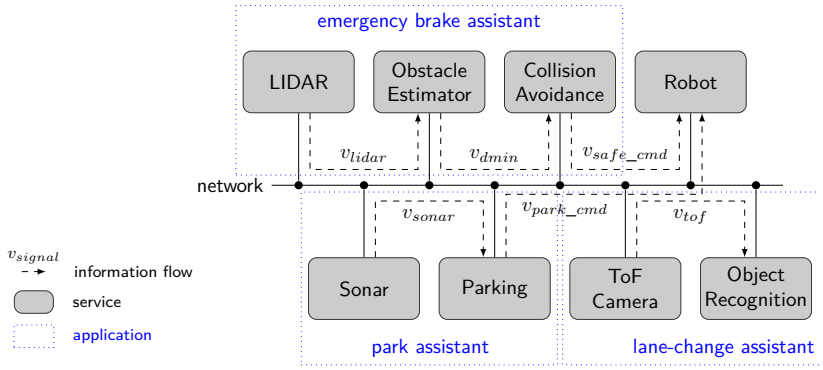
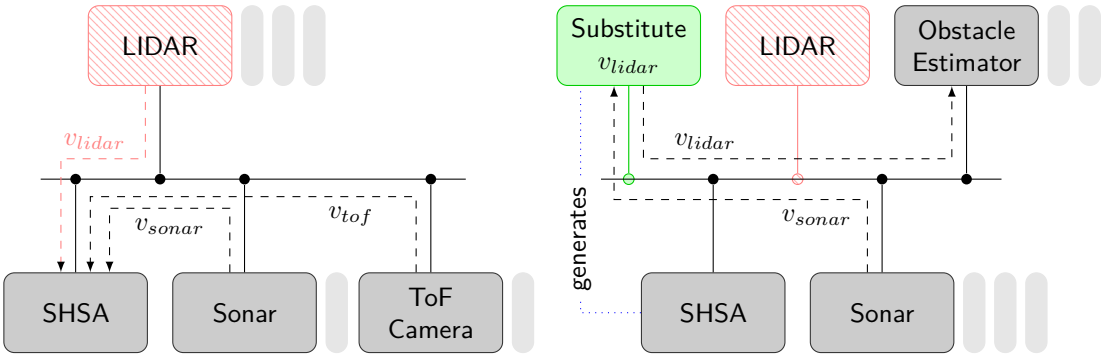


Figure 1.9: Mobile robot's CPS applications.

The three applications operate independently from each other. Yet, all three use information about the distance to obstacles, though using different sensors (optimized for the specific application). Such information redundancy is modeled in a knowledge base exploited by SHSA to detect and recover from failures. For instance, the SHSA service connected to the communication network detects when the LIDAR fails by comparing

the LIDAR's outputs against the one of the sonar and camera (Fig. 1.10a). In case of a LIDAR failure, SHSA generates a substitute which provides the necessary information, here derived from the outputs of the sonar, to the controllers (Fig. 1.10b).



(a) SHSA fault detection.

(b) SHSA recovery.

Figure 1.10: Failure scenario in the mobile robot (only relevant services are fully shown).

Building Blocks of SHSA

SHSA is no one-shot mechanism, it continuously observes the CPS and adapts to failures. The self-adaptation community distinguishes following parts of the control loop for adaptation [12]: *i)* collect information about the environment and derive internal system properties, *ii)* analyze the observations, *iii)* decide about and plan the adaptation, *iv)* and finally act. Self-healing is typically split into detection, diagnosis (find the root cause) and recovery of a failure (i.e., summarizes the plan and act step).

SHSA in particular, monitors and detects failures of observation services (e.g., sensors, filters or state estimators) which manifest as faulty information in the network. On failure of an observation service, the fault detection unit triggers the recovery which subsequently generates a substitute publishing the information or signal previously provided by the failed observation service. Both, the fault detection and recovery unit, exploit a knowledge base encoding redundancy of signals in the communication network. This thesis develops and improves the highlighted blocks and actions in Figure 1.11.

1.6.3 Architecture for Structural Adaptation

In contrast to parametric adaptation [12] which changes the *behavior* of a service itself, SHSA adapts the system's structure. However, most CPSs are designed to be static, i.e., their architecture does not provide means to add, remove or reconnect software services. As a consequence, SHSA has to be enabled and facilitated not only by a dedicated SHSA service (a piece of software) but also by the underlying platform comprising hardware, system software and middleware software.

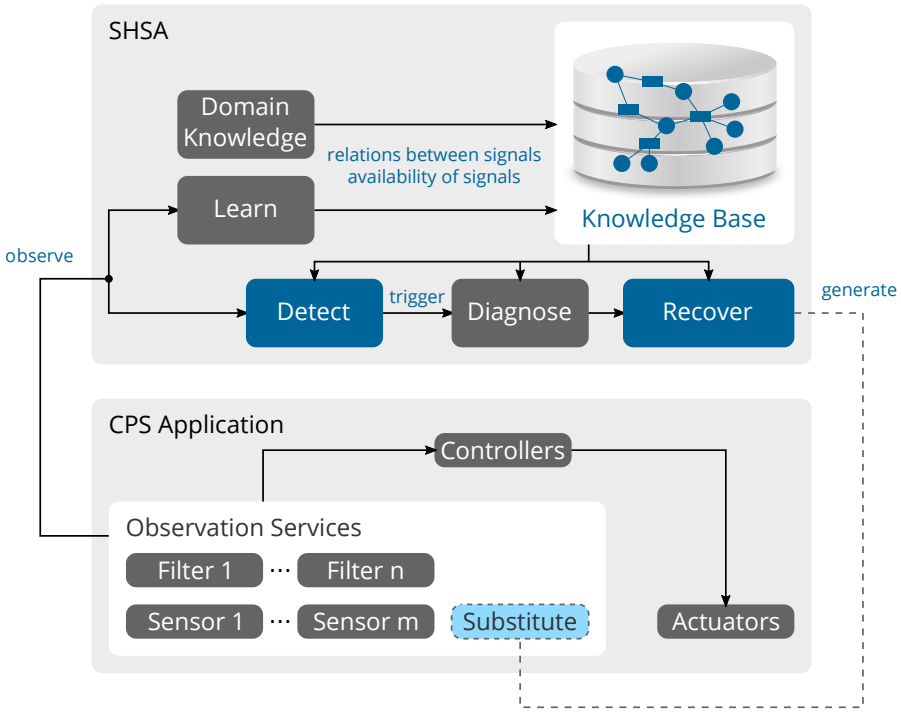


Figure 1.11: Building blocks or services (boxes) and information flow (arrows) of SHSA.

System Model

This work focuses on the adaptation in the software cyber-part of a CPS (in contrast to hardware reconfiguration or dynamic reconfiguration of FPGAs). SHSA considers physical components or hardware which comprise at least one software component (e.g., the driver of a sensor), only. The software components implement services (e.g., application or SHSA) and run on a platform typically including hardware, operating system and the middleware. The latter enables the communication between services (Fig. 1.12).

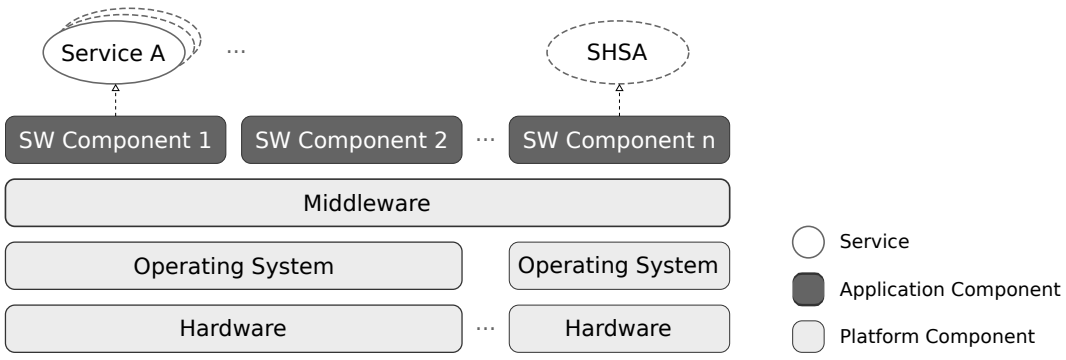


Figure 1.12: Typical architecture of a distributed CPS.

Components are used to describe the hierarchical structure of a system, mainly to cope with the complexity of a system. The behavior of the system is often described by services and their interactions. For instance, the Service-Oriented Architecture (SOA) [40] completely omits the physical structure. Abstracting away the hierarchical structure of the system usually simplifies the design and implementation of application and management services. For instance, an IoT application developer does not need to cope with the possibly heterogeneous capabilities of the devices itself or know *how* the information is provided, but gathers and processes the information necessary to implement a specific service. Similarly, the SHSA algorithm focuses on *what* information is available or needed to detect and recover from a failure.

The following paragraph states the architectural requirements – that are the needs to be fulfilled by the platform – to apply structural adaptation in general. Subsequently, keeping the requirements in mind, suitable implementations are proposed. Last but not least, the generation of a substitute and its interface is discussed.

Requirements

- *Dynamic Composability:* The system shall be dynamically composable out of services while preventing side effects or undesired emergent behavior.

  Self-healing systems need this property to, e.g., *i)* disconnect or shutdown failed services, *ii)* setup (start and connect) new services (e.g., substitutes for failed services), *iii)* adapt the information flow, or *iv)* migrate services.

  - *Reconfigurable Information Flow:* The information flow shall be reconfigurable.

    Services consume and/or produce information. When a service is added, removed or changed the communication to and from the service must be established.

  - *Common Communication Interface:* The adaptation mechanism and application services should be connected to a common communication interface.

    Distributed systems are typically assembled out of several subsystems often provided by different manufacturers or developed for different applications with different demands (e.g., performance vs. real-time). As a consequence, the communication network might differ within a system and between different subsystems. The reconfiguration of the information flow is easier or less complicated when a common communication interface or Application Programming Interface (API) is provided. When no such interface is available such layer of abstraction has to be developed (in the form of, e.g., a middleware or gateway).

  - *Freedom of Interference:* The adaptation shall not compromise the system's functionality.

    This is needed to ensure the consistency of the system before, during and after self-adaptation, e.g., when a new component is integrated. For instance, when

adding a component the timeliness of a real-time system shall not be violated. Violation of this requirement can lead to (subsequent) failures.

– *Fault Containment:* A failure of a service shall not propagate.

Components usually trust their inputs, i.e., have no monitor for inputs installed (cf. runtime enforcement [17]). Hence, failures may propagate through the communication subsystem, reach the physical system via actuators and cascade back to the cyber-system via sensors. Therefore the system shall provide means to stop failures propagating, e.g., by a fault detection mechanism that is able to shutdown a failed service.

- *Information Access:* The system shall provide means to collect information about the system's state.

  Information about the cyber- and physical system is necessary to define when and what adaptation should be performed. To this end, the system shall provide a common communication interface and enable simple means for gathering information relevant to the system's state.

Design of a Platform Enabling Structural Adaptation

The service-oriented approach is a good candidate to hide the hierarchy, subsystem ownership and the heterogeneity of the devices [25, 55, 40]. Management components (e.g., service registry, orchestration, authorization) and a middleware enable the services to provide and consume other services. State-of-the-art IoT infrastructures or middlewares adopt SOA (e.g., OPC-UA [42], DDS [45], CoAP [54], MQTT [23, 39]). A middleware provides a common communication interface and easy information access in the form of an API. DDS [45] for instance, provides also means to ensure the desired performance or QoS requirements regarding the communication (cf. freedom of interference). For updated and additional services (e.g., substitutes) CPU time and memory have to be reserved and monitored.

Using data-centric publish-subscribe communication simplifies the reconfiguration of information flow [41, 45]. The receiver or consumer *subscribes* to the information of interest. The sender of a specific message type is unknown or irrelevant to the receiver. A dedicated management component handles the communication channels w.r.t. to provided and consumed information. For instance, when the provider of some information changes, the consumer of the same information will automatically be reconnected to the new provider (e.g., a SHSA substitute).

Fault containment can be achieved by a fault detection unit within subsystems or at the interfaces of components. Monitors within a subsystem should have the ability to suppress faulty information or shutdown failed services.

Template for Arbitrary Services

For the substitute, SHSA uses a service template which can be configured at start-up to execute a desired function.

Services of a distributed application process some inputs to generate some output. For instance, a substitute for a failed information takes related information as input, transforms these inputs by a relation function and publishes the result as output. A generic service implements $P : o = f(I)$ and therefore has three parameters: *i)* an input vector I, *ii)* a function f implemented as a program or code P, and *iii)* an output o. In this work a single output was sufficient, however, w.l.o.g. the output can be extended to a vector. Listing 1 depicts the pseudo code of a generic service.

Algorithm 1 Generic Service

```
1: procedure SERVICE(I, P, o)
2:     Inputs ← { InputInterface(i) | i ∈ I }      ▷ install interface (e.g., subscribe)
3:     Output ← OutputInterface(o)
4:     while True do
5:         x ← Inputs.READ()                        ▷ get current input values
6:         y ← Evaluate(P, x)                       ▷ pass inputs through function
7:         Output.WRITE(y)                          ▷ publish result
8:     end while
9: end procedure
```

SHSA requires dynamic composition of services, i.e., the substitute and its program is generated during runtime. It is therefore convenient to express the function f in an interpretable language. In the presented prototypes, the SHSA service generates the code P – a string representing Python code – during runtime and starts the service similarly to the following example:

```
$ service -i /input1/data /input2/data -p "y = x[0] + x[1]" -o /output/data
```

Listing 1.1: Run a custom service. The example service subscribes to two inputs and publishes its sum to the output.

The implementation of the interface (*read* and *write* in Listing 1) depends on the architecture or middleware used. Typically, the used framework provides an API for the interface to the communication network. Nevertheless, receiving information of different sources raises challenges in the *read* function, especially in asynchronous communication networks. These challenges and possible solutions for the input interface are outlined in the next section.

Temporal Alignment in Asynchronous Communication

Data can be transmitted in a continuous stream (synchronous), scheduled to be sent and received at specific points in time (time-triggered), transmitted upon events or sent in irregular intervals (asynchronous). In event-driven or asynchronous communication networks the exact point in time of the reception of a message is unknown.

Services like sensor fusion units, majority voters or a substitute generated by SHSA has to combine information from different sources. The interface of such services therefore has to handle possibly asynchronous, multi-rate measurements and nondeterministic, best-effort communication of the IoT or CPS network (Fig. 1.13). To this end, the services have to perform a so-called *temporal alignment* of the inputs before the computation can be performed.

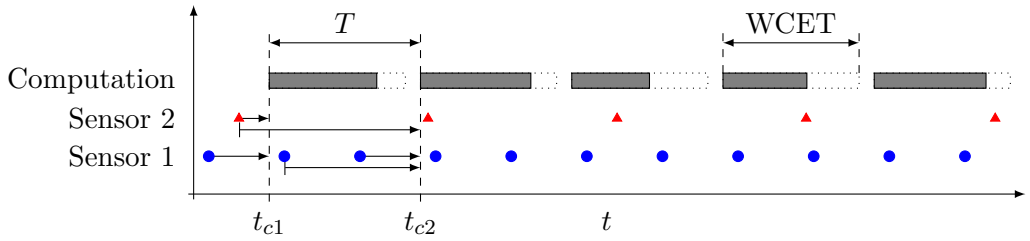


Figure 1.13: Typical timeline of a periodically executed computation and its asynchronous and multi-rate inputs (sensor 1 and 2). Received inputs are aligned to the computation's starting point (indicated as arrows from the observations received towards the first two computations starting at t_{c1} and t_{c2}).

The following factors contribute to the way temporal alignment has to be performed:

- Single-rate vs. multi-rate.

  When inputs arrive with the same rate the computation can be triggered as soon as all measurements are received. In the ideal case, all measurements are received at the same point in time, that is synchronously. In asynchronous communication networks the reception time of the samples (from different sources) differ and may suffer from jitter and delays.

  When sensors have different sample rates (cf. 10ms and 1s of sensors in a mobile robot) and all inputs are awaited before a computation is performed, the result can be inaccurate. Moreover, a state estimation might not be able to follow the dynamics of the system. For instance, the localization of a mobile robot might lack behind when using data from a camera with a low frame rate.

- Jitter, delay or loss.

  Inputs may be sampled with the same rate but the time between two measurements may vary around the sampling period. The variance of the sampling period is called *jitter* and can be caused by clock drifts or scheduling policies. Moreover, the

message may be delayed due to a requested re-transmission of a packet or re-routing or even get lost.

- Worst-Case Execution Time (WCET) of the computation.

  Some computations may need a significant amount of CPU time (e.g., particle filtering for localization, path planning) that might exceed a sensor's sample period.

The timeline of an observation (Fig. 1.14) distinguishes between the time t_x, when the CPS variable actually adopted the value, the timestamp of the measurement t_s included in the message (typically the time when the message has been generated and provided by the sensor) and the time the message is received t_r or used t_c. Unfortunately, t_x is unknown and typically does not match t_s (though the difference is often neglectable).

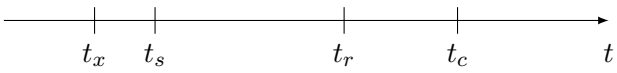


Figure 1.14: Timeline of an observation.

When the delay of an observation is high ($t_c \gg t_x$) the computation uses outdated information. Outdated information can lead to a poor accuracy of the computation. For instance, the older the measurements the worse is the state estimate of a Kalman filter. In real-time systems, outdated information is faulty information [30]. For instance, an engine speed controller may escalate when the measured and received rotational speed does not match the actual one.

It is therefore desired to execute the computation as close to t_x as possible, e.g., as soon as a measurement is received, and as often as possible, e.g., each time an input is received. Two solutions are proposed: *ii)* a so-called *port* collecting inputs for a computation, or *i)* state estimators to forward estimate the inputs.

Port. Using a port, the computation simply takes the latest samples as input (Fig. 1.13: latest observations of sensor 1 and sensor 2). The port comprises a slot for each input. At the beginning the port is empty. When an input arrives, the corresponding slot is filled with the received value. As soon as the port is full, i.e., a value of each input has been received, the computation can be executed. Subsequent receptions overwrite the corresponding slot of the port, such that the computation always uses the latest input values. The computation is triggered periodically.

State Estimation. State estimation (e.g., a Kalman filter) calculates the expected value of a CPS variable at t_c given a state transition model, the evidence observed (received measurements) and the timestamps of the measurements (time passed). Each observation, considering its timestamp, can be used to correct or smooth an existing estimate [52]. Moreover, a state estimator can compensate delayed and missing measurements by forward estimation. This is particularly useful when fusing multi-rate sensors (some with probably low sample rates). It is therefore more accurate than the port.

The port is suitable for resource-constrained devices where state estimation is not feasible. State estimation requires more computational resources than the port, a state transition model and a decent clock synchronization (for reliable timestamps). Choosing T as small as possible proved to increase the accuracy (Ch. 4). However, the WCET of the computation limits its execution rate ($T > WCET$). State estimators can be combined with a port to collect inputs from high-rate sensors (for sensors with a sample period smaller than the estimation's WCET).

1.6.4 Knowledge Base of Redundancy

The SHSA knowledge base comprises

- the relations between CPS variables each expressed as a function $\mathsf{r} : \mathsf{v}_o = f(\mathsf{V}_I)$ to compute an output variable v_o from a set of input variables V_I,
- the availability of signals and their mapping to variables,
- and optionally custom properties of variables, relations and signals to measure the performance of such.

The knowledge base is typically setup by a domain expert or learned (Ch. 3). A starting point to setup a knowledge base is an information flow graph of the application. For instance, Figure 1.9 depicts the services and information flow implemented on the mobile robot we've used as SHSA prototype. Relating the signals to each other, while combining signals providing the same variable (e.g., distance to obstacles), gives the knowledge base in Figure 1.15.

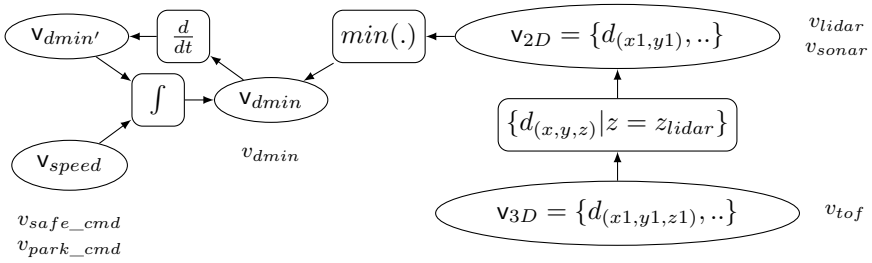


Figure 1.15: The knowledge base of the mobile robot exploitable for a self-healing collision avoidance. Boxes are relations. Ellipses are variables. Variables are annotated with available signals (from Figure 1.9).

The available signals v_{signal} are collected from the information flow in Figure 1.9. Some signals capture the same kind of information, e.g., the CPS variable v_{2D} – the set of distance measurements to obstacles around the robot at a fixed height – which is provided by v_{lidar} and v_{sonar}. The minimum distance v_{dmin} can be derived by taking the $min(.)$ of observations from v_{2D} (in the application performed by the service "Obstacle Estimator").

The ToF camera provides observations v_{tof} in the form of 3D depth images v_{3D}. The 2D distance measurements can be extracted from this image, by selecting the observations at the height of the LIDAR. The last observation of the minimum distance $\mathsf{v}_{dmin'}$ may be saved and integrated by the speed of the robot v_{speed} (including the direction) to estimate the current minimum distance v_{dmin}.

Structure

The knowledge base $K = (\mathsf{V}, \mathsf{R}, E)$ is a bipartite directed graph (which may also contain cycles) with independent sets of variables V and of relations R of a CPS. V and R are the nodes of the graph. Edges E specify the input/output interface of a relation. In particular, v_i is an input variable for r iff $\exists(\mathsf{v}_i, \mathsf{r}) \in E$. v_o is the output variable of r iff $\exists(\mathsf{r}, \mathsf{v}_o) \in E$.

Substitution

The SHSA recovery substitutes a needed but failed signal v_{failed} by semantically equivalent signals (a signal v is *needed* when v is input to a controller, e.g., v_{dmin} in above example). To this end, the failed signal is first mapped to the corresponding variable v_{sink} in the knowledge base. SHSA then checks the knowledge base for redundancy. If the variable is not directly provided by (another) signal, SHSA searches for a possible substitution.

A substitution concatenates several functions to relate variables to each other. In particular, the substitution s of v_{sink} is a connected acyclic sub-graph of the knowledge base with the following properties: *i)* The output variable is the only sink of the substitution. *ii)* Each variable has zero or one relation as predecessor. *iii)* All input variables of a relation must be included (it follows that the sources of the substitution graph are variables only).

A substitution s is *valid* if all source variables are provided by signals, otherwise the substitution is *invalid*. The set of valid substitutions of a variable v is referred to as $S(\mathsf{v})$. Only a valid substitution can be instantiated (e.g., to a substitute for a failed signal) by concatenating the relations to a function which takes selected signals as input.

For instance, the signal from the ToF camera v_{tof} can be used to derive the minimum distance to an obstacle v_{dmin} (Fig. 1.16).

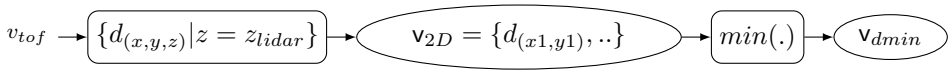


Figure 1.16: A valid substitution for v_{dmin}.

Properties and Utilities

Additional properties are used to distinguish and assess valid substitutions. For instance, a signal is *accurate*, or a relation is *costly* (e.g., consumes more or less processing time). A property p_i is expressed by a value.

Substitutions are evaluated by a utility function, that is, a substitution is ranked based on its utility u_s which is a function of the properties $u(p_1, .., p_p)$ of its elements (variables, relations, input signals).

The optimal or best substitution $s_{best} = \arg\max_{s \in S(\mathsf{v}_{sink})} u_s$ of a set of valid substitutions $S(\mathsf{v}_{sink})$ for a variable v_{sink} is the substitution with the highest utility.

Implementation

The knowledge base has been encoded as a graph (as described above) and as a Prolog program, that is a set of facts (e.g., relations) and rules (e.g., how to find a valid substitution). The implementation in a graph library takes advantage of optimized search algorithms and has been used to speed up the recovery process (Ch. 6). A short implementation in Prolog has been used to setup the monitor detecting faults by plausibility checks of redundant signals (Ch. 5). Because the *setup* of the monitor typically has no timing constraints, the runtime of the substitution search is neglectable.

An essential requirement of SHSA is dynamic composability of the system (Sec. 1.6.3). For instance, substitutes are generated and connected to the system during runtime. The prototypes therefore use an interpreter to run a substitute. In particular, the function of a relation is written in Python.

Note, that the knowledge base is also adaptive, i.e., it might change during runtime (structure, availability of signals, etc.).

1.6.5 Fault Detection

The information redundancy can be used to perform plausibility checks on signals (Ch. 5). A query on the knowledge base can identify related information. However, the signals need to be compared in a common domain, i.e., the signals have to be transferred through relations such that the outputs represent the same variable (Fig. 1.17). This section describes how to setup a monitor given the knowledge base of redundancy and how to compare related observations over time.

Setup

The relations and availability of signals are part of the knowledge base. Given a signal under test, all its valid substitutions are searched. The monitor subscribes to the input signals used by the substitutions. On adaptations (e.g., change in the availability of signals due to updates or addition/removal of components) the setup can be re-run.

Observation Model

Signals representing CPS variables – that are physical entities – over time, are prone to uncertainties in space (e.g., noise) and time (e.g., delays).

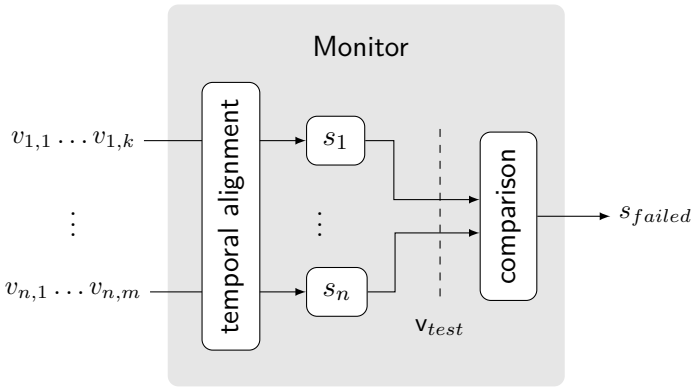


Figure 1.17: Abstract representation of a monitor service. Input signals $v_{i,j}$ from the communication network are aligned and transferred through the substitution s_i to a common domain (the variable of interest v_{test}). The outputs of the substitutions are compared and the substitution with the highest error (if there is any) returned.

A direct comparison of the output values of the monitor's substitutions would indicate a fault (e.g., floating point numbers are rarely exactly equal). Interval arithmetic provides simple means to express the confidence (cf. probability distribution expressed by particles) into the value and timestamp of a signal's snapshot – an observation (see also Fig. 1.18):

$$v(t_s) = (\boldsymbol{v}, \boldsymbol{t}) \qquad \boldsymbol{v} = [\underline{v}, \overline{v}] \qquad \boldsymbol{t} = [\underline{t}, \overline{t}] \quad t_s \in [\underline{t}, \overline{t}] \tag{1.1}$$

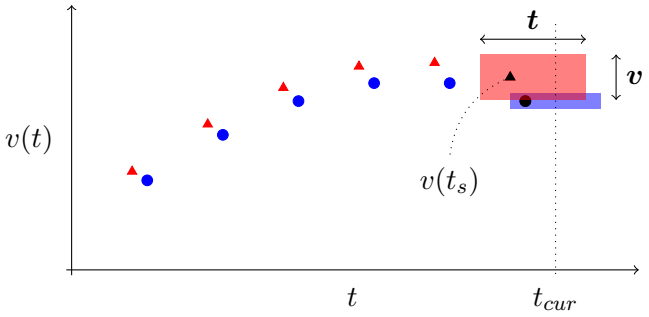


Figure 1.18: Two one-dimensional signals (observations over time). A marker represents an observation: the measured value and timestamp. The confidence regions of the latest observations are highlighted.

Observations of the same variable (here, the output variable of the substitutions which is the variable under test) can be compared if the time intervals overlap. Non-overlapping of the value intervals of two observations to compare indicates an error. For instance,

the latest observations in Figure 1.18 are *comparable* (i.e., time intervals overlap) and *match* (i.e., value intervals overlap).

Fault Detection by Comparison

The monitor executes the plausibility check periodically. On each call, the monitor compares the observations collected since the last monitor call and a (configurable sized) buffer of observations of previous executions. The buffer is used to compensate late receptions or high communication delays (temporal alignment in Fig. 1.17).

One monitor call consists of several steps:

- New and buffered observations are *collected* into combinations of inputs for the substitutions of the monitor, such that their time intervals overlap.
- Each combination of inputs is *executed*, i.e., the inputs' value intervals are transferred through the corresponding substitution's relations by applying interval arithmetic. The time interval of the output is the intersection of the inputs' time intervals (the time the output is considered valid).
- Next, the monitor *pairwise compares* the outputs of the last step. The error per substitution is aggregated and ranked.
- The monitor returns the substitution with the highest error, or none if the errors are all 0. A subsequent moving average or moving median filter can compensate outliers of input signals.

The result of the monitor – failed substitution s_{failed} – is used to trigger the recovery. In addition, a root-cause analysis is necessary to identify the failed input signal that lead to a faulty output of s_{failed}. For simplicity, in the presented examples, the substitution had only a single input to be able to omit the diagnosis. Moreover, the current implementation assumes that the inputs of the substitutions are independent of each other.

1.6.6 Recovery

The recovery of a failed signal is triggered by a monitor, e.g., a simple watchdog given a timeout for a signal or the fault detection presented in the last section. The recovery unit searches for a valid substitution and instantiates a substitute (based on the node template from Sec. 1.6.3). The substitute provides the previously failed signal by transferring related signals through a valid substitution (Fig. 1.19).

The knowledge base can be searched, e.g., via Depth-First Search (DFS) [22], or queried for valid substitutions (Ch. 5). Yet, the setup and start-up of a substitute (instantiation and connection) may take significantly more time than the search itself (Ch. 3). It is therefore desired to search for the best substitution right away.

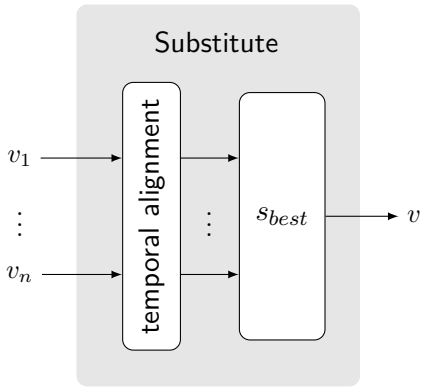


Figure 1.19: Abstract representation of a substitute. Input signals v_i are aligned and transferred through the substitution s_i.

Speed-up Substitution Search

Guided-search, assessing each substitution with a utility (Sec. 1.6.4), decreases the runtime of the search for the best recovery option. Moreover, the proposed algorithm – Self-Healing by Property-Guided Structural Adaptation (SH-PGSA) (Ch. 6) – can be split into threads which then can be executed in parallel.

First the failed signal is mapped to its variable, say v_{sink}, in the knowledge base. v_{sink} is the root of each possible substitution tree. Initially, SH-PGSA creates a worker (cf. thread) to substitute v_{sink}. The worker discovers the relations connected to v_{sink} (Fig. 1.20 a: two relations, i.e., two possibilities to proceed). It selects the best relation and returns additional workers for other possibilities. Each worker represents another possible substitution for v_{sink}. The workers also maintain the utility of its substitution. The worker with the highest utility is allowed to proceed for one step, that is, it checks the next connected relations of unprovided variables (Fig. 1.20 b: e.g., $W[0]$). Note, that all variables of a chosen relation have to be available or must be substituted, subsequently. When the utility of a substitution is monotonically decreasing (by addition of a relation, or variable as a consequence) the optimal (w.r.t. utility function) substitution is returned first. A worker may also abort, when the substitution is invalid and unprovided variables cannot be further substituted. The interested reader is referred to Chapter 6 for the detailed algorithm.

Verify Structural Adaptation

Chapter 7 adds runtime safety assurance to SHSA on the example of Ontology-based Runtime Reconfiguration (ORR) [22] to ensure safe self-adaptation. Amorim *et al.*defines Multidirectional Conditional Safety Certificates (ConSerts M) [1], that is a language to describe certificates or safety contracts between interacting application (horizontal) and platform (vertical) services. The certificates are created per service during design time, however, can be assessed during runtime, e.g., at system integration or on changes.

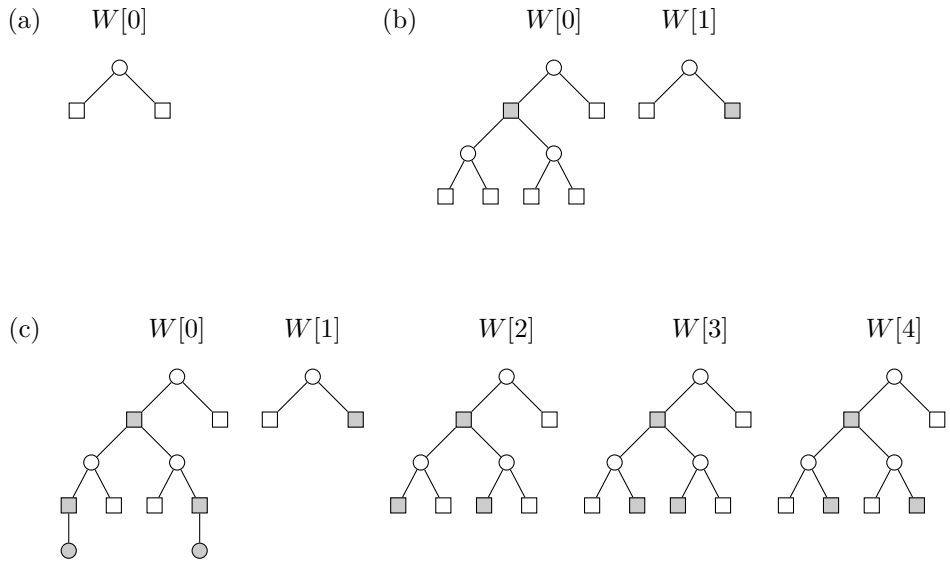


Figure 1.20: Visualization of workers and its underlying substitutions (encoding: provided variables and selected relations are filled). (a) Initial set of workers that is a single worker starting from the sink node. (b) The worker $W[0]$ selects the best option (here: left relation), however, returns worker $W[1]$ for the other relation and possible substitution. $W[0]$ chooses the left relation which leads to two unprovided variables which must be substituted by subsequent relations. (c) $W[0]$ selects the best combination of relations and returns additional three workers for other possible combinations.

ConSerts M. A certificate describes the guarantees and demands of a service. The guarantees characterize the quality properties which may rely on demands, e.g., of the platform or other services. Each guarantee or demand consists of a statement (e.g., a potential failure mode, or required processing resources) and a level of confidence described by a Safety Integrity Level (SIL) based on an industry standard (e.g., IEC 61508). For instance, a sensor in a vehicle guarantees to be of SIL level B, while a controller demands this sensor to have a specific level, e.g., at least A, and requires the platform to provide sufficient processing resources to ensure deadlines.

Runtime Safety Assurance. An adaptation must not violate the certificates. Hence, before the actual recovery (here, installation of the substitute) the safety contracts are re-evaluated.

1.7 Scientific Contribution

This work contributes to the problem statement (Sec. 1.2) and answers the research questions (RQ; Sec. 1.3) as follows.

Contribution 1: *Self-healing challenges and possibilities of CPSs (RQ 1).*

Chapter 2 collects the attributes of a resilient system and analyzes the faults and threats in modern CPSs. Moreover, it surveys approaches to fault detection and recovery which extends existing surveys on self-healing for CPSs. Chapter 2 also lists the main challenges for these methods to be applied in CPSs possibly connected to the IoT, notably, *i)* limited knowledge of device behavior and interface semantics (e.g., to retrieve a reference behavior for anomaly detection or to substitute failed services), *ii)* long-term dependability and security, that is to ensure resilience also after environmental, functional or technological changes of the system, and *iii)* adaptation, verification, validation and robustness of the resilience techniques.

Contribution 2: *Architectural requirements and design guidelines for SHSA (RQ 2/2.1).*

Chapter 3 elaborates on the requirements and possible designs of an adaptive architecture. It provides guidelines for system designers and engineers on how to enable structural adaptation. This chapter also demonstrates the applicability of SHSA in an automotive use case. It outlines the code generation of the substitute component given a service template and the integration of the substitute into the system.

Contribution 3: *Interface for CPS components - temporal alignment of inputs (RQ 2.2).*

Chapter 4 proposes an interface for components collecting inputs, e.g., sensor measurements, in an unreliable communication network like the IoT. On the example of sensor fusion this chapter shows how to handle asynchronous, multi-rate and/or delayed inputs. This interface is used for generated substitutes too (Ch. 3).

Contribution 4: *Formalization and extension of the redundancy model.*

This thesis enhances the work of Höftberger [22] by formalizing the (runtime) model of information redundancy in the CPS communication, providing means to evaluate a possible substitution (Ch. 6) and enabling adaptation of the knowledge base (Ch. 5).

Contribution 5: *Fault detection based on implicit redundancy (RQ 3).*

The recovery is triggered by a monitor exploiting the very same knowledge base to identify related information or signals. The monitor uses a simple observation model, which considers uncertainties in space and time of the communicated information, to compare related information (Ch. 5).

Contribution 6: *Performance measure for substitutions (RQ 5) and speed-up of the substitution search, i.e., the planning step of the recovery (RQ 4).*

Chapter 6 introduces a performance measure for substitutions (a user-defined utility function). This enables the comparison of substitutions. Moreover, this chapter presents

SH-PGSA – a guided-search algorithm which decreases the average execution time of the substitution search and returns the best substitution first.

Contribution 7: *Demonstration of runtime assurance of the planned recovery (RQ 5).*

Chapter 7 showcases a method to verify a planned recovery or valid substitution, once SH-PGSA returns. For instance, the planned substitute component must follow the safety contracts of its connected components.

1.8 Conclusion

This work introduces SHSA which provides means to exploit redundancy during runtime. SHSA sets the information or signals, communicated in a CPS, into relation. A knowledge base – created by domain experts and/or learned during runtime – collects these relations, i.e., the functions between signals. The knowledge base can be searched for relations to detect faults by comparing signals or to recover failed signals.

Compared to traditional explicit redundancy where replicas of critical components are added during design time, SHSA creates minor additional costs (for running the monitor and recovery service) given that some implicit redundancy exists. The proposed approach seems to be a promising candidate to overcome many challenges of resilient CPSs (Ch. 2):

- *Resource limitations:* The knowledge base of redundancy can be scaled to the capabilities of a system or can be split or distributed to subsystems or to a single component.
- *Heterogeneity and complexity:* SHSA does not require inside knowledge or reconfiguration capabilities of probably heterogeneous and black-box components of the system. It acts on the common communication interface, i.e., the network of the CPSs (Ch. 3-4).
- *Real-time constraints:* The knowledge base is rule-based (Ch. 5), i.e., the relations and available signals clearly define the valid substitutions. Moreover, the knowledge base can be kept static (constant relations and properties) to ensure deterministic adaptations.
- *Long-term dependability and security (by self-healing):* SHSA can handle various fault scenarios – even some which have not been considered during design time.
- *Robustness of resilience techniques and scalability (by adaptation):* The knowledge base can be distributed or adapted during runtime by updating, adding or removing relations between variables and changing the availability of information (Ch. 5-6).
- *Validation and verification (towards guaranteeing resilience):* Substitutions can be evaluated or ranked (Ch. 6), and planned adaptations can be verified (Ch. 7).

1.9 Future Work

Though SHSA has been successfully applied in several use cases, some challenges remain and new research opportunities have been uncovered. Especially for market-readiness, a proper methodology of creating the knowledge base has to be developed. Moreover, the verification of self-adaptive systems is a big challenge itself (cf. safety-critical systems – note that the presented implementations are prototypes only and cannot be applied directly, e.g., in a self-driving car).

Verify various types of system or substitute requirements. Other requirements than safety certificates have to be verified before self-adaptation, e.g., non-interference, desired accuracy, deadline or rate constraints of substitutes. Also to cope with the probably diverse QoS requirements of different applications in an IoT.

Learn the structure of the knowledge base. Chapter 3 shows how to learn particular relations, here by regression. Another necessary part is to learn the structure of the knowledge base to further automate self-healing. A starting point may be techniques from related knowledge bases, e.g., dependency graphs or Bayesian networks [52].

Possible extensions and improvements to the presented solution are:

Generalize the knowledge base using ontologies. The current implementation focuses on the semantic relations between outputs of observation services in CPSs that are physical entities or CPS variables. The prototypes also handle the differences in the syntax of the signals (e.g., the message structure capturing the observations). The SHSA knowledge base may be generalized for different kind of services, not limited to CPS observation components. For instance, Höftberger's ontology can be improved with the presented extensions (e.g., properties, requirements) and recent IoT ontologies, to build a common description language for service interfaces which enable self-healing.

Enhance substitution search by monitor observations. In future work the monitor can be extended to evaluate used substitutions over time. Recall, the implemented fault detection uses all possible substitutions to compare related signals. Once, the monitor detects a failure (mismatching outputs of the substitutions), the used substitutions and its differences or errors over time can be forwarded to the recovery or reconfiguration unit. The error over time is another property of a substitution which should contribute to the substitution's utility. The substitution search would get obsolete in this case. It would be replaced by a re-evaluation of the utility of each substitution used by the monitor.

Selection of substitutions for fault detection. In case there are too many substitutions available for a signal, the monitor can (or has to) limit the number of

substitutions (to check with a reasonable rate), e.g., by evaluating substitution utilities. Furthermore, it is desired to select a diverse group of substitutions to enable fault diagnosis, i.e., identify the faulty input (recall: only the outputs of the substitutions are comparable; however, one of the inputs will trigger the failure).

Include verification of requirements in the substitution search. The verification step is put after searching the best substitution but before running the substitute to ensure safety assurance of the system. However, the verification may be part of the substitution search. For instance, the rules of ConSerts M or other requirements could be encoded in Prolog and thus considered during the search.

Related approaches that are not discussed within this work are fault prevention (keep the system healthy, e.g., by avoiding failures or predictive maintenance), fault diagnosis (find the source of the fault), and maintenance (cf. offline recovery). Nevertheless, implicit redundancy can be exploited in this area too. For instance, a service may use a relevant part of the knowledge base to compare its input against available substitutions to avoid an error (cf. runtime enforcement [17]).

1.10 Bibliography

[1] T. Amorim, A. Ruiz, C. Dropmann, and D. Schneider. Multidirectional Modular Conditional Safety Certificates. In F. Koornneef and C. van Gulijk, editors, *Computer Safety, Reliability, and Security*, Lecture Notes in Computer Science, pages 357–368. Springer International Publishing, 2015.

[2] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. on Dependable and Secure Computing*, 1(1):11–33, 2004.

[4] E. Bartocci and Y. Falcone, editors. *Lectures on Runtime Verification: Introductory and Advanced Topics*. Programming and Software Engineering. Springer International Publishing, 2018.

[5] A. Beugnard, J.-M. Jézéquel, and N. Plouzeau. Contract aware components, 10 years after. *arXiv preprint arXiv:1010.2822*, 2010.

[6] R. Bloomfield, K. Netkachova, and R. Stroud. Security-Informed Safety: If It's Not Secure, It's Not Safe. In A. Gorbenko, A. Romanovsky, and V. Kharchenko, editors, *Software Engineering for Resilient Systems*, pages 17–32, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[7] M. Botts and A. Robin. OGC ® SensorML: Model and XML Encoding Standard. Standard OGC 12-000, Open Geospatial Consortium, Feb. 2014.

[8] A. L. Buczak and E. Guven. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys Tutorials*, 18(2):1153–1176, Secondquarter 2016.

[9] I. Butun, S. D. Morgera, and R. Sankar. A Survey of Intrusion Detection Systems in Wireless Sensor Networks. *IEEE Communications Surveys Tutorials*, 16(1):266–282, First 2014.

[10] Center for Machine Learning and Intelligent Systems, University of California, Irvine. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/index.php.

[11] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[12] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy,

M. Tivoli, D. Weyns, and J. Whittle. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, Lecture Notes in Computer Science, pages 1–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[13] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J. P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In R. de Lemos, H. Giese, H. A. Müller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, Lecture Notes in Computer Science, pages 1–32. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[14] N. Dragoni, A. Giaretta, and M. Mazzara. The Internet of Hackable Things. In *Proceedings of 5th International Conference in Software Engineering for Defence Applications*, pages 129–140, Cham, 2018. Springer International Publishing.

[15] J. Dürrwang, M. Rumez, J. Braun, and R. Kriesten. Security Hardening with Plausibility Checks for Automotive ECUs. In *VEHICULAR 2017: The Sixth International Conference on Advances in Vehicular Systems, Technologies and Applications*, pages 38–41. IARIA, July 2017.

[16] Y. Falcone, L. Mariani, A. Rollet, and S. Saha. Runtime Failure Prevention and Reaction. In *Lectures on Runtime Verification*, volume 10457 of *Lecture Notes in Computer Science*, pages 103–134. Springer, Feb. 2018.

[17] Y. Falcone, L. Mounier, J.-C. Fernandez, and J.-L. Richier. Runtime enforcement monitors: Composition, synthesis, and enforcement abilities. *Formal Methods in System Design*, 38(3):223–262, 2011.

[18] D. Ghosh, R. Sharman, H. Raghav Rao, and S. Upadhyaya. Self-healing systems — survey and synthesis. *Decision Support Systems*, 42(4):2164–2185, Jan. 2007.

[19] M. Gröndahl, K. Collins, and J. Glanz. The Dangerous Flaws in Boeing's Automated System. *The New York Times*, Mar. 2019.

[20] T. A. Henzinger, M. Minea, and V. Prabhu. Assume-guarantee reasoning for hierarchical hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 275–290. Springer, 2001.

[21] T. Herault and Y. Robert, editors. *Fault-Tolerance Techniques for High-Performance Computing*. Computer Communications and Networks. Springer International Publishing, 2015.

[22] O. Höftberger. *Knowledge-Based Dynamic Reconfiguration for Embedded Real-Time Systems*. PhD thesis, TU Wien, Institute of Computer Engineering, Wien, 2015.

[23] U. Hunkeler, H. L. Truong, and A. Stanford-Clark. MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks. In *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, pages 791–798, Jan. 2008.

[24] R. Isermann. *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Springer, Berlin Heidelberg, 2006.

[25] ISO. ISO/IEC 18384:2016 Information technology - Reference Architecture for Service Oriented Architecture (SOA RA) – Part 1: Terminology and Concepts for SOA. https://collaboration.opengroup.org/projects/soa/documents.php?action=show&dcat=&gdid=36358, Nov. 2016.

[26] Kaggle. Datasets | Kaggle. https://www.kaggle.com/datasets.

[27] S. Kehrer, O. Kleineberg, and D. Heffernan. A comparison of fault-tolerance concepts for IEEE 802.1 Time Sensitive Networks (TSN). In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–8, Sept. 2014.

[28] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan. 2003.

[29] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[30] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer, New York, 2nd edition, 2011.

[31] S. Kounev, P. Lewis, K. L. Bellman, N. Bencomo, J. Camara, A. Diaconescu, L. Esterle, K. Geihs, H. Giese, S. Götz, P. Inverardi, J. O. Kephart, and A. Zisman. The Notion of Self-aware Computing. In S. Kounev, J. O. Kephart, A. Milenkoski, and X. Zhu, editors, *Self-Aware Computing Systems*, pages 3–16. Springer International Publishing, Cham, 2017.

[32] J. Laprie. Resilience for the Scalability of Dependability. In *Fourth IEEE International Symposium on Network Computing and Applications*, pages 5–6, July 2005.

[33] J.-C. Laprie. From dependability to resilience. In *38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*, pages G8–G9. Citeseer, 2008.

[34] E. A. Lee and S. A. Seshia. An Introductory Textbook on Cyber-physical Systems. In *Proceedings of the 2010 Workshop on Embedded Systems Education*, WESE '10, pages 1:1–1:6, New York, NY, USA, 2010. ACM.

[35] P. R. Lewis, A. Chandra, S. Parsons, E. Robinson, K. Glette, R. Bahsoon, J. Torresen, and X. Yao. A Survey of Self-Awareness and Its Application in Computing Systems. In *2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pages 102–107, Oct. 2011.

[36] H. B. Mitchell. *Multi-Sensor Data Fusion: An Introduction.* Springer, Berlin Heidelberg, 2007.

[37] R. Mitchell and I.-R. Chen. A Survey of Intrusion Detection Techniques for Cyber-physical Systems. *ACM Comput. Surv.*, 46(4):55:1–55:29, Mar. 2014.

[38] S. Ntalampiras. Detection of Integrity Attacks in Cyber-Physical Critical Infrastructures Using Ensemble Modeling. *IEEE Transactions on Industrial Informatics*, 11(1):104–111, Feb. 2015.

[39] OASIS. MQTT Version 3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[40] OASIS. OASIS SOA Reference Model. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm#overview.

[41] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The Information Bus: An Architecture for Extensible Distributed Systems. In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, SOSP '93, pages 58–68, New York, NY, USA, 1993. ACM.

[42] OPC Foundation. OPC Unified Architecture (OPC UA) Specification. https://opcfoundation.org/developer-tools/specifications-unified-architecture.

[43] Open Robotics. ROS2 Overview. https://index.ros.org/doc/ros2/, June 2019.

[44] S. Papa, W. Casper, and S. Nair. A transfer function based intrusion detection system for SCADA systems. In *2012 IEEE Conference on Technologies for Homeland Security (HST)*, pages 93–98, Nov. 2012.

[45] G. Pardo-Castellote. OMG Data-Distribution Service: Architectural overview. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pages 200–206, May 2003.

[46] S. Poledna. *Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism.* Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[47] H. Psaier and S. Dustdar. A survey on self-healing systems: Approaches and systems. *Computing*, 91(1):43–73, Jan. 2011.

[48] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: An open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, volume 3, page 5. Kobe, Japan, 2009.

[49] R. Rajkumar. A Cyber–Physical Future. *Proceedings of the IEEE*, 100(Special Centennial Issue):1309–1312, May 2012.

[50] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems: The next computing revolution. In *Design Automation Conference*, pages 731–736, June 2010.

[51] J. Rushby. Runtime certification. In *International Workshop on Runtime Verification*, pages 21–35. Springer, 2008.

[52] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 3rd edition, 2016.

[53] D. C. Schmidt, J. White, and C. D. Gill. Elastic Infrastructure to Support Computing Clouds for Large-Scale Cyber-Physical Systems. In *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 56–63, June 2014.

[54] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). 2014.

[55] The Open Group. SOA Features and Benefits. In *SOA Source Book*. 7 edition, Aug. 2011.

[56] S. Thrun, W. Burgard, D. Fox, and R. C. Arkin. *Probabilistic Robotics*. MIT Press, Aug. 2005.

[57] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. S. Jubert, M. Mazura, M. Harrison, M. Eisenhauer, et al. Internet of things strategic research roadmap. *Internet of Things - Global Technological and Societal Trends*, 1(2011):9–52, 2011.

[58] K. Vipin and S. A. Fahmy. FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications. *ACM Comput. Surv.*, 51(4):72:1–72:39, July 2018.

[59] D. Weyns. Software Engineering of Self-Adaptive Systems: An Organised Tour and Future Challenges. Springer, 2017.

[60] M. Wollschlaeger, T. Sauter, and J. Jasperneite. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE Industrial Electronics Magazine*, 11(1):17–27, 2017.

[61] I. Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. A. Ahmed, A. Gani, M. Imran, and M. Guizani. Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges. *IEEE Wireless Communications*, 24(3):10–16, June 2017.

CHAPTER 2

A Roadmap Toward the Resilient Internet of Things for Cyber-Physical Systems

Abstract. The Internet of Things (IoT) is a ubiquitous system connecting many different devices – the *things* – which can be accessed from the distance. The cyber-physical systems (CPS) monitor and control the things from the distance. As a result, the concepts of dependability and security get deeply intertwined. The increasing level of dynamicity, heterogeneity, and complexity adds to the system's vulnerability, and challenges its ability to react to faults. This paper summarizes state-of-the-art of existing work on anomaly detection, fault-tolerance and self-healing, and adds a number of other methods applicable to achieve resilience in an IoT. We particularly focus on non-intrusive methods ensuring data integrity in the network. Furthermore, this paper presents the main challenges in building a resilient IoT for CPS which is crucial in the era of *smart CPS* with enhanced connectivity (an excellent example of such a system is connected autonomous vehicles). It further summarizes our solutions, work-in-progress and future work to this topic to enable "Trustworthy IoT for CPS". Finally, this framework is illustrated on a selected use case: A smart sensor infrastructure in the transport domain.

Keywords. Anomaly Detection, Cyber-Physical Systems (CPS), Internet of Things (IoT), Monitoring, Resilience, Long-Term Dependability and Security, Self-Adaptation, Self-Healing.

The content of this chapter has been published: *D. Ratasich, F. Khalid, F. Geissler, R. Grosu, M. Shafique, and E. Bartocci. A Roadmap Toward the Resilient Internet of Things for Cyber-Physical Systems. IEEE Access, 7(1):13260–13283, Jan. 2019.* DOI: 10.1109/ACCESS.2019.2891969.

CHAPTER 3

A Self-Healing Framework for Building Resilient Cyber-Physical Systems

Abstract. Self-healing is an increasingly popular approach to ensure resiliency, that is, a proper adaptation to failures and attacks, in cyber-physical systems (CPS). A very promising way of achieving self-healing is through structural adaptation (SHSA), by adding and removing components, or even by changing their interaction, at runtime. SHSA has to be enabled and supported by the underlying platform, in order to minimize undesired interference during components exchange and to reduce the complexity of the application components. In this paper, we discuss architectural requirements and design decisions which enable SHSA in CPS. We propose a platform that facilitates structural adaptation and demonstrate its capabilities on an example from the automotive domain: a fault-tolerant system that estimates the state-of-charge (SoC) of the battery. The SHSA support of the SoC estimator is enhanced through the existence of an ontology, capturing the interrelations among the components and using this information at runtime for reconfiguration. Finally, we demonstrate the efficiency of our SHSA framework by deploying it in a real-world CPS prototype of a rover under sensor failure.

The content of this chapter has been published: *D. Ratasich, O. Höftberger, H. Isakovic, M. Shafique, and R. Grosu. A Self-Healing Framework for Building Resilient Cyber-Physical Systems. In 2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC), pages 133–140, May 2017.* DOI: 10.1109/ISORC.2017.7.

CHAPTER 4

Generic Sensor Fusion Package for ROS

Abstract. Sensor fusion combines multiple sensor measurements to improve a controller's knowledge about the internal state of an observed physical environment. Many such sensor fusion techniques exist and have been implemented for the Robot Operating System (ROS). However, they often have been developed for specific applications and cannot be easily reused for other applications. Reasons are the use of application-specific, partly undocumented interfaces, and the often limited reconfigurability caused by a tight coupling of the implementation to an application-specific purpose. Our approach is based on the concept of a fusion node which provides a configurable sensor fusion service with a generic interface. Fusion nodes can be interconnected to combine several sensor fusion techniques, can be attached to any single-dimension value sensor, can handle asynchronous multi-rate measurements and are robust regarding indeterministic, best-effort communication. This paper presents, to the best of our knowledge, the first generic sensor fusion package (GSFP) for ROS which collects various exemplary sensor fusion methods implemented as fusion nodes. We demonstrate the feasibility of our package in a small test application. Main benefits of our contribution are the developed ROS package's independence regarding specific sensors or applications, the easy integration of configurable fusion nodes in existing applications, and the composition of fusion nodes to realize complex sensor fusion scenarios.

The content of this chapter has been published: *D. Ratasich, B. Frömel, O. Höftberger and R. Grosu. Generic Sensor Fusion Package for ROS. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 286–291, Sept. 2015.* DOI: 10.1109/IROS.2015.7353387.

CHAPTER 5

Adaptive Fault Detection Exploiting Redundancy with Uncertainties in Space and Time

Abstract. The Internet of Things (IoT) connects millions of devices of different cyber-physical systems (CPSs) providing the CPSs additional (implicit) redundancy during runtime. However, the increasing level of dynamicity, heterogeneity, and complexity adds to the system's vulnerability, and challenges its ability to react to faults. Self-healing is an increasingly popular approach for ensuring resilience, that is, a proper monitoring and recovery, in CPSs. This work encodes and searches an adaptive knowledge base in Prolog/ProbLog that models relations among system variables given that certain implicit redundancy exists in the system. We exploit the redundancy represented in our knowledge base to generate adaptive runtime monitors which compare related signals by considering uncertainties in space and time. This enables the comparison of uncertain, asynchronous, multi-rate and delayed measurements. The monitor is used to trigger the recovery process of a self-healing mechanism. We demonstrate our approach by deploying it in a real-world CPS prototype of a rover whose sensors are susceptible to failure.

Keywords. Fault Detection, Self-Healing, Cyber-Physical Systems, Redundancy Model, Observation Model

The content of this chapter has been published: *D. Ratasich, M. Platzer, R. Grosu, E. Bartocci. Adaptive Fault Detection Exploiting Redundancy with Uncertainties in Space and Time. In 2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), pages 23–32, June 2019.* DOI: 10.1109/SASO.2019.00013.

CHAPTER 6

Self-Healing by Property-Guided Structural Adaptation

Abstract. Self-healing is an increasingly popular approach ensuring resiliency, that is, a proper adaptation to failures, in cyber-physical systems (CPS). A very promising way of achieving self-healing is through structural adaptation (SHSA), by replacing a failed component with a substitute component. We present a knowledge base modeling relations among system variables given that certain implicit redundancy exists in the system and show how to extract a substitute from that knowledge base using guided search. The result of our search, i.e., the substitute, is optimal w.r.t. a user-defined utility function considering properties of the system variables (e.g., accuracy). We demonstrate our approach - Self-Healing by Property-Guided Structural Adaptation (SH-PGSA) - by deploying it in a real-world CPS prototype of a rover whose sensors are susceptible to failure. We further show the increased runtime performance to find the optimal substitute by comparing it to related work.

The content of this chapter has been published: *D. Ratasich, T. Preindl, K. Selyunin, and R. Grosu. Self-healing by property-guided structural adaptation. In 2018 IEEE Industrial Cyber-Physical Systems (ICPS), pages 199–205, May 2018.* DOI: 10.1109/ICPHYS.2018.8387659.

CHAPTER 7

Runtime Safety Assurance for Adaptive Cyber-Physical Systems: ConSerts M and Ontology-Based Runtime Reconfiguration Applied to an Automotive Case Study

Abstract. Cyber-Physical Systems (CPS) provide their functionality by the interaction of various subsystems. CPS usually operate in uncertain environments and are often safety-critical. The constituent systems are developed by different stakeholders, who – in most cases – cannot fully know the composing parts at development time. Furthermore, a CPS may reconfigure itself during runtime, for instance in order to adapt to current needs or to handle failures. The information needed for safety assurance is only available at composition or reconfiguration time. To tackle this assurance issue, the authors propose a set of contracts to describe components' safety attributes. The contracts are used to verify the safety robustness of the parts and build a safety case at runtime. The approach is applied to a use case in the automotive domain to illustrate the concepts. In particular, the authors demonstrate safety assurance at upgrade and reconfiguration on the example of ontology-based runtime reconfiguration (ORR). ORR substitutes a failed service by exploiting the implicit redundancy of a system.

The content of this chapter has been published: *T. Amorim, D. Ratasich, G. Macher, A. Ruiz, D. Schneider, M. Driussi and R. Grosu. Runtime Safety Assurance for Adaptive Cyber-Physical Systems: ConSerts M and Ontology-Based Runtime Reconfiguration Applied to an Automotive Case Study. In N. Druml, A. Genser, A. Krieg, M. Menghin and A. Hoeller (Eds.), Solutions for Cyber-Physical Systems Ubiquity, pages 137–168, IGI Global, 2018.* DOI: 10.4018/978-1-5225-2845-6.ch006.

List of Figures

List of Tables

List of Algorithms

Acronyms

AC Autonomic Computing. 6

AI Artificial Intelligence. 1, 2, 6, 7, 16

API Application Programming Interface. 21–23

ConSerts M Multidirectional Conditional Safety Certificates. 31, 36

CPS Cyber-Physical System. 1–11, 13, 14, 16–20, 24–28, 33–35, 55

DFS Depth-First Search. 5, 30

IoT Internet of Things. 1–5, 7, 8, 10, 11, 13–17, 21, 22, 24, 33, 35

LIDAR Light Detection and Ranging. 18, 19, 27

MAPE-K Monitor, Analyze, Plan and Execute using a Knowledge base. 2, 6, 10, 55

ML Machine Learning. 16

ORR Ontology-based Runtime Reconfiguration. 12, 31

QoS Quality-of-Service. 16, 22

ROS Robot Operating System. 6, 8, 9

SH-PGSA Self-Healing by Property-Guided Structural Adaptation. 31, 34

SHSA Self-Healing by Structural Adaptation. 3–14, 17–24, 26–28, 31, 33–35, 55

SIL Safety Integrity Level. 32

SOA Service-Oriented Architecture. 21, 22

TMR Triple Modular Redundancy. 5, 8

ToF Time-of-Flight. 18, 27

TSN Time-Synchronized Network. 8

WCET Worst-Case Execution Time. 25, 26

Denise Ratasich

Curriculum Vitae

Education

2014–Present **PhD**, *TU Wien*, Vienna.
Topic: *Self-healing cyber-physical systems*
Designing and implementing self-healing cyber-physical systems that adapt to implicit redundancy on sensor failures during runtime (demonstrator: mobile robot).

2011–2014 **Master of Science**, *TU Wien*, Vienna, in Computer Engineering, with distinction.
Master thesis: *Generic low-level sensor fusion framework for cyber-physical systems*
Investigation of various state estimation methods, temporal alignment of asynchronous multi-rate measurements and implementation of an application-independent sensor fusion package for the Robot Operating System (demonstrator: position estimation on a mobile robot).

2008–2011 **Bachelor of Science**, *TU Wien*, Vienna, in Computer Engineering, with distinction.
Bachelor thesis: *Energy harvesting* and *Embedded control of a rotating LED-display*

2003–2008 **High school diploma**, *HTL (polytechnic)*, Pinkafeld, in Electronics, with distinction.

Experience

2019–Present **Data Scientist**, *ÖBB - Business Competence Center*, Vienna.

Develop, maintain and manage the data ingestion and storage of ÖBB's IoT infrastructure. Perform various predictions given available data.

2014–2019 **Teaching and Research Assistant**, *TU Wien*, Vienna.

Organization of several courses (one of these a bachelor course with >400 students). Give introductory lectures (e.g., programming in C, operating systems and POSIX). Develop, prepare and grade (lab) exercises (C, Matlab, Python, Robotics, Probabilistic Models). Prepare computer tests and hold exams. Supervise bachelor and master theses.

Research in the field of sensor fusion, self-healing, machine-learning, runtime monitoring and CPS. Design and implementation of self-healing systems within European and national projects. Integration into demonstrators with partners (TTTech, VIF Graz, AVL Graz, TNO, Fraunhofer IESE).

Continuous training via university courses, programmes (TUtheTOP), invited talks and participation in conferences in the field of machine learning, self-organizing systems, artificial intelligence, robotics, cyber-physical systems, worst-case execution time analysis and runtime monitoring.

Skills

Soft Skills	Team player, always curious and eager to learn something new, sympathetic, self-initiative, organisational talent, flexible
Engineering	Software- and Hardware-Development, Test-Driven Development, Automate Processes (Scripting), Sensor Fusion, Machine Learning, Artificial Intelligence, Robotics, Embedded Programming, (Operating) System Programming, WCET-Analysis, System Modelling and Analysis, Automation
Programming	languages among others: C, C++, Python, Prolog, Bash, Lua, Java, LaTeX
Tools	make/CMake, Robot Operating System, Docker, git, scikit-learn, pandas, numpy, Matlab, Eclipse, emacs, Org mode, LibreOffice, MS Office, WordPress
OS	Linux, Windows 7/8/10

Languages

German	Mothertongue	
English	Intermediate	*Conversationally fluent*
Croatian	Basic	*Basic words and phrases*

Interests

- (Beach-)Volleyball
- Open source projects and tools
- Automate processes
- Zero Waste
- Woodworking
- Gardening
- Hiking

Publications

(all peer-reviewed)

First-Author

2019 Adaptive Fault Detection exploiting Redundancy with Uncertainties in Space and Time. *In Proc.: International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*

2019 A Roadmap Toward the Resilient Internet of Things for Cyber-Physical Systems. *In Journal: IEEE Access*

2018 Self-Healing by Property-Guided Structural Adaptation. *In Proc.: International Conference on Industrial Cyber-Physical Systems (ICPS)*

2017 A Self-Healing Framework for Building Resilient Cyber-Physical Systems. *In Proc.: International Symposium on Real-Time Computing (ISORC)*

2015 Generic Sensor Fusion Package for ROS. *In Proc.: International Conference on Intelligent Robots and Systems (IROS)*

Co-Author

2018 CPS/IoT Ecosystem: A platform for research and education. *In Proc.: Workshop on Embedded and Cyber-Physical Systems Education (WESE)*

2018 Runtime Safety Assurance for Adaptive Cyber-Physical Systems: ConSerts M and Ontology-Based Runtime Reconfiguration Applied to an Automotive Case Study. *In Book: Solutions for Cyber-Physical Systems Ubiquity*

2017 A Survey of Hardware Technologies for Mixed-Critical Integration Explored in the Project EMC^2. *In Proc.: International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*

2017 Computing with Biophysical and Hardware-efficient Neural Models. *In Proc.: International Work-Conference on Artificial and Natural Neural Networks (WANN)*

2015 Neural Programming: Towards Adaptive Control in Cyber-Physical Systems. *In Proc.: Conference on Decision and Control (CDC)*

2015 Collision Avoidance for Mobile Robots with Limited Sensing and Limited Information About the Environment. *In Proc.: International Conference on Runtime Verification (RV)*

References and Links

TU Wien Prof. Ezio Bartocci, TU Wien Project Leader in IoT4CPS
Phone: +43 1 58801 18226

LinkedIn https://www.linkedin.com/in/denise-ratasich-6708b689/

ResearchGate https://www.researchgate.net/profile/Denise_Ratasich