Technische Universität München
TUM School of Computation, Information and Technology

On Engineering Self-Adaptive Cyber-Physical Systems

Ana Petrovska

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung einer

Doktorin der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz: Prof. Matthias Grabmair, Ph.D.

Prüfer*innen der Dissertation:

1. Prof. Dr. Alexander Pretschner
2. Assoc. Prof. Raffaela Mirandola, Ph.D.,
   Politecnico di Milano

Die Dissertation wurde am 17.11.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 03.04.2023 angenommen.

Acknowledgments

"To do things right, first you need love, then technique."

Antonio Gaudi

Working on this thesis was a journey to finding myself. I will be eternally grateful for all that I have been given throughout. However, this thesis would not have been possible without the support of many people. This thesis is because of you and for you.

First, I want to thank my supervisor Professor Alexander Pretschner. Alex, I am deeply grateful for being given this chance to join your group—a chance that has completely changed my life. Thank you for believing in me, constantly challenging me, pushing me, and guiding me. For always asking the most difficult questions that no one else dares to ask, and trusting that I am capable in finding the answers. We might not have all the answers yet, but the depth that this thesis went is beyond everything I ever thought I would be able to achieve. It was an honor for me to be able to work with you and to grow under your supervision. I also want to thank Professor Raffaela Mirandola. You agreeing to be my second examiner has more meaning to me than you will ever know.

Special gratitude goes to Professor Ilias Gerostathopoulos and Professor Stefan Kugele. You both have had an essential part in my research and overall journey. Ilias, my mentor, thank you for the passion and energy that you brought in the early stages of my work. Our paths, including my decision to research self-adaptive systems, have miraculously overlapped during your stay at TUM, and you have been someone I have always looked up to. And, Stefan, there are no words to thank you enough for everything that you have done for me. You are one of the very few people I could always rely on and a constant source of strength and stability in everything I did. Our countless hours of discussion about my research at any time of the day, your critical thinking, attention to detail, and expertise on the formal aspects of the work, in combination with your patience and tranquillity, played an immense role for me. Thank you for believing in me and seeing the value in my work, even in the moments when I was struggling to see it myself. I also want to say thanks to Wolfgang Böhm, Professor Manfred Broy, Professor Danny Weyns, Diego Marmsoler and Florian Grigoleit with whom I had chance to establish different collaborations.

To all my past and present colleagues in the chair, especially to Thomas, Stephan, Severin, Florian, Patrick, Amjad, Valentin, Mojdeh and Traudl: thank you! Without being able to share the pain with you and without your presence in the chair, I am not sure if I would have been able to survive this journey and endure until the end. Without a doubt, our group has some of the smartest people I have met, and I am grateful for the chance to be your colleague and to learn from all of you.

This thesis would not be what it is without all the wonderful students that I supervised over the years who contributed in unique ways towards shaping my research, often complementing my thoughts and ideas with different and fresh perspectives: Shahin, Sergio, Julian, Malte, Dmitrij, Sebastian, Muhammad Ansab, Tamer, Theo, Adrian, Andreas, Nour, Naum, Guan, and a couple of other students from the SSACPS praktikum. Thank

you for choosing me to be your supervisor and bringing your individuality and authenticity to our collaboration. I truly appreciate all of you, and being your supervisor was one of the best and most fun part of my work.

My professional life and purpose are only complete if my quest to be continuously challenged as an engineer and a researcher is accompanied by fighting and advocating for diversity and better representation of women in Computer Science. Therefore, my Ph.D. experience would have been partial without the Women in CS@TUM family: Professor Anne Brüggeman-Klein and everyone on the team, some of whom have become dear friends of mine over the years. Having the freedom to grow and shape this community and to lead the most amazing, inspiring, courageous, and fearless group of women (and some men) was and still is one of the biggest passions in my life. The spirit of this group is unbeatable, and I am beyond thankful to have had a chance to work with all of you.

And finally, I want to thank my friends and family. Sirma, Ivana and Sandra, thank you for always being by my side to listen and console me empathetically. I am sorry that most of the conversations and topics that we had for the last couple of years, from my side, boiled down to my work and research. And to Sara and Nadine, I feel our friendships reached new depths since we could understand each other how pursuing a Ph.D. feels when no one else could understand us. Mama and tato, you are both my rock, and I cannot imagine a world in which this thesis could happen without you. I will be forever grateful for the unconditional love and, more importantly, the unconditional support you have given to me throughout this whole time. The last couple of years have brought us closer than ever before, and I love you more than words can describe. I also want to say thanks to my extended family: to my grandma, all my aunts, uncles, cousins, nieces, and nephews, for always showing care about my well-being and the progress of my thesis. And lastly, I want to thank Thomas and Monika. Since the time we met, our hearts and souls have beat together, and I am so grateful to have you in my life.

Zusammenfassung

Die Idee der selbst-adaptiven Systeme und allgemein der selbst-* Systeme wurde in der Literatur mit der Veröffentlichung des berühmten Papiers *The vision of autonomic computing*, allgemein bekannt als IBM-Manifest, vor fast zwei Jahrzehnten eingeführt und etabliert. Seitdem hat die Popularität von selbst-adaptiven Systemen in verschiedenen Forschungsbereichen erheblich zugenommen. Trotz der umfangreichen Arbeiten in diesem Forschungsbereich und der lebhaften Forschungsgemeinschaft fehlt es in der wissenschaftlichen Literatur jedoch immer noch an einem genaueren Verständnis von selbst-adaptiven Systemen, insbesondere an klaren Spezifikationen und nutzbaren, praktikablen Definitionen. Bislang wird die Terminologie im Zusammenhang mit selbst-adaptiven Systemen noch sehr ungenau verwendet, was dazu führen kann, dass verschiedene Akteure ein unterschiedliches Verständnis von diesen Systemen haben. Eine mehrdeutige Beschreibung der Terminologie und das Fehlen eines präzisen Verständnisses von selbst-adaptiven Systemen machen ein ohnehin schon komplexes Forschungsgebiet noch komplizierter und hindern die Forschergemeinschaft daran, ihre Ziele in vollem Umfang zu verwirklichen. Für mehr Klarheit und Kohärenz—die möglicherweise zu einer noch größeren Bedeutung dieses Fachgebiets führen—halten wir es für unerlässlich, eine gemeinsame Terminologie festzulegen und ein besseres Gesamtverständnis für die Semantik dieser Terminologie zu erlangen. Das übergeordnete Ziel dieser Dissertation ist es daher, ein gemeinsames Verständnis dieser Systeme zu erarbeiten und zu vertiefen.

Diese Dissertation besteht aus zwei Teilen: *theoretische Grundlagen selbst-adaptiver Systeme* und *dem Engineering selbst-adaptiver cyber-physischer Systeme (CPS) in dynamischem Kontext.* Als Beitrag im ersten Teil dieser Dissertation führen wir Grundlagenforschung durch, die sich vor allem mit der Frage beschäftigt, wie sich selbst-adaptive Systeme von "gewöhnlichen", nicht-adaptiven Systemen unterscheiden. Konkret schlagen wir im ersten Teil zunächst eine Taxonomie autonomer Systeme vor und führen eine systematische Literaturrecherche darüber durch, wie selbst-adaptive Systeme in der Literatur definiert werden. Einige Erkenntnisse aus der vorgeschlagenen Taxonomie autonomer Systeme und die Einschränkungen der bestehenden formalen Definitionen selbst-adaptiver Systeme dienen als Grundlage für den Rest unserer theoretischen Beiträge. In unseren weiteren theoretischen Beiträgen erörtern wir, dass das Verständnis der Systemanpassung für eine nachfolgende Definition von selbst-adaptiven Systemen unabdingbar ist, und daraufhin definieren wir Systemadaption formell. Wir schlagen auch einen Bewertungsrahmen vor, der notwendig ist, um zu beurteilen, ob ein System adaptiv ist, und der im Entwicklungsprozess selbst-adaptiver Systeme berücksichtigt werden muss.

Im zweiten Teil dieser Dissertation führen wir mehr angewandte Forschung durch, in der wir uns auf die Entwicklung von selbst-adaptiven CPS konzentrieren, die in sich verändernden, dynamischen und teilweise beobachtbaren Kontexten operieren. Im Zusammenhang mit unseren theoretischen Beiträgen schlagen wir eine logische Architektur für die Entwicklung dieser besonderen Art von selbst-adaptiven Systemen vor, gefolgt von einer Methodik für eine domänen- und systemunabhängige Modellierung des Wissens in der Adaptionslogik,

die Wissensaggregation und Schlussfolgerungen unter Unsicherheiten während der Laufzeit ermöglicht.

Durch die Kombination dieser beiden Teile bieten wir einen kontinuierlichen und einheitlichen Entwicklungsprozess von formalen Definitionen über eine logische Architektur, die in das vorgeschlagene theoretische Rahmenwerk eingebettet ist, bis hin zu einer konkreten Systemimplementierung aus dem Gebiet der Multi-Agenten-Robotik. Wir erreichen dies, indem wir eine Brücke zwischen den verschiedenen Arten von Beiträgen und Lösungen bilden, die alle eine unterschiedliche Generalität und Abstraktionsebene aufweisen.

Abstract

The idea of self-adaptive systems, and in general self-* systems, was initiated and established in the literature with the publishing of the famous paper on *The vision of autonomic computing*, broadly known as IBM manifesto almost two decades ago. Since then, the popularity of self-adaptive systems has significantly increased across various research domains. However, despite the extensive work in this research field and the vibrant research community, the literature still lacks a more precise understanding of self-adaptive systems, including clear specifications and usable, working definitions. So far, the terminology related to self-adaptive systems is still used liberally, which can result in different parties having a different understanding of these systems. An ambiguous description of the terminology and the lack of a precise understanding of self-adaptive systems add complexity to an already complex field of research and hinder the community from endeavouring to a fuller extent. For improved clarity and coherence—potentially yielding to even higher prominence of this domain—we consider as essential to set a common terminology and gain a better comprehension of the semantics of that terminology. As a result, the overarching objective of this dissertation is to broaden and set a common understanding of these systems.

This thesis contains of two parts: *theoretical foundations for self-adaptive systems* and *on engineering self-adaptive cyber-physical systems (CPSs) in dynamic context.* In the first part of this thesis, we conduct foundational research, mainly motivated by how self-adaptive systems differ from the "ordinary," non-adaptive systems. Concretely, as contributions in the first part, we first propose a taxonomy of autonomous systems and conduct a systematic literature review on how self-adaptive systems are defined in the literature. Some findings from the proposed taxonomy of autonomous systems and the limitations of the existing formal definitions of self-adaptive systems serve as a basis for the rest of our theoretical contributions. In the rest of the theoretical contributions, we discuss that understanding system adaptation is essential for a subsequent definition of self-adaptive systems, and in response, we define system adaptation formally. We also propose a framing necessary to debate whether a system is adaptive, which needs to be considered in the engineering process of self-adaptive systems, and lastly, we define self-adaptive systems.

In the second part of this thesis, we conduct more applied research, in which we focus on the engineering of self-adaptive CPSs that operate in changing, dynamic, and partially observable contexts. Within our theoretical contributions, we propose a logical architecture for engineering this particular type of self-adaptive system, followed up with a methodology for a domain- and system-independent modeling of the knowledge in the adaptation logic, which enables knowledge aggregation and reasoning under uncertainties during run-time.

By combining both of these parts, we provide a continuous and streamlined engineering process from formal definitions to a logical architecture embedded in the proposed theoretical framework to a concrete system implementation from the domain of multi-agent robotics. We achieve this by bridging the various types of contributions and solutions, all of them with different generality and level of abstraction.

Contents

Part I

Introduction and Background

1 Introduction

This chapter presents an introduction to the topic and the fundamental problems addressed by this thesis. It discusses the goals, the research gaps, the solutions, and the contributions of this work.

1.1 Self-Adaptive Cyber-Physical Systems: Motivation, Challenges and Shortcomings

Traditionally, an important and considerable part of the research in software engineering focuses on handling software complexity and software quality, in particular attaining specific quality properties [107]. Woods [126] has also presented his perspective on how systems evolve from monolithic architectures back in the 1980s, to intelligent, connected, real-time, complex systems in the 2020s that are comprised of many interconnected software and hardware components. Therefore, the importance of attaining the systems quality and the systems functional goals increases proportionally with the increasing complexity of the software systems, which also implies increased costs for managing those systems.

In recent years there has been an expanding interest in dealing with software complexity and software quality at run-time or operation time after the systems have been initially deployed. Four main factors have contributed towards this.

Firstly, there are various new emerging systems and technologies, including 5G, Internet of Things (IoT), and Cyber-Physical Systems (CPS), with a higher demand for embedded, mobile, and ubiquitous applications. *Secondly*, these modern systems introduce new levels of complexity. Namely, the systems are dynamic themselves and often need to operate in highly dynamic and unanticipated environments with continuously changing conditions that introduce uncertainties, e. g., unanticipated resource availability, dynamically changing system goals, and different hardware and software faults and failures. Furthermore, related to this point, the engineering of these modern and dynamic systems goes beyond the system itself. Instead, the context in which the system is intended to operate needs to be considered—with almost equal importance—in the engineering process of the systems. This phenomenon, engineering dynamic systems also in correspondence to their context, requires a significant paradigm shift in most of the currently established engineering practices. *Thirdly*, the systems' development process itself has become more dynamic and agile, aiming to deploy the systems as early as possible, even sometimes "too early." This results in engineers not being able to anticipate everything during the design phase of the systems and, therefore, to incomplete requirements during the requirements elicitation phase. *Lastly*, there are aspects of the system and its environment that simply cannot be anticipated during the design time of the system, regardless of the time available, which unavoidably lead us towards building imperfect and flawed systems. For example, engineers will always have partial knowledge about the dynamic and changing environments (or contexts) in which the systems will be operating, similarly as some of the internal system

faults or failures that occur during run-time. Thus, uncertainty is an intrinsic property of every modern and dynamic system. By definition uncertainty refers to situations involving incomplete or inconsistent information such that it is not possible for the system to know which environmental or system state hold at a specific point in time [104].

As a result of all of the above, the idea behind systems that can handle changes and uncertainties autonomously [116], called self-adaptive systems, has emerged in the literature in the past ten to twenty years. "In a world where computing systems rapidly converge into large open ecosystems, uncertainty is becoming the de-facto reality of most systems we build today, and it will be a dominating element of any system we will build in the future. The challenges software engineers face to tame uncertainty are huge. Self-adaptation has an enormous potential to tackle many of these challenges" [118].

1.1.1 Cyber-Physical Systems and Run-Time Uncertainties

The world is changing, and so are systems. In recent years, the widespread availability of cost-effective embedded systems with increasing computation power and the expansion of wireless networks have lead to a solid foundation for the emergence and advancement of the omnipresent Cyber-Physical Systems (CPSs) in a multitude of different domains, with continuously growing socio-economic influence. Namely, CPSs are software-intensive systems embedded in the physical world to monitor, control, and coordinate various processes in both the physical and the digital world. Also, contemporary CPSs are often mobile and operate autonomously. Autonomous refers to the ability of the CPSs to operate in an uncontrolled environment without the need for any human or electro-mechanical guidance [4, 75]. Additionally, multiple CPSs can also communicate with each other and collaboratively work together towards achieving one or several common objectives. Thus, multiple collaborating CPSs can provide shared and more complex functionalities that a single system in isolation cannot attain. Furthermore, since the CPSs themselves are composed of many interacting and interconnected components, they inherit all the complexities of modern large-scale distributed systems [90]. As a result, the engineering of CPSs must consider the dynamic composition of their internal structural components, as well in some situations, the dynamic composition of groups of CPSs. In summary, the prominent research in the field of CPSs—considered the second generation of embedded systems—involves modelling, designing, developing, and analysing networked, real-time, distributed systems. As a consequence of their deployment into and their interaction with the physical world, they operate in dynamic, changing and uncertain environments, and come to close interactions with humans. The subset of the environment relevant for the CPS, which interacts with the system is called *operational* or *execution context* [102, 86, 32, 90]. In general, context is the system-relevant part of the environment that influences the system's input, behaviour and state but cannot be influenced by the developers of the system, and it should be accepted as given.

The business continuity of the modern CPSs requires these systems to operate efficiently, correctly and reliably despite being exposed to a variety of uncertainties during their operation. In dynamic systems, e. g., CPSs, sources of uncertainty occur in one of the following three phases: requirements, design and run-time phase [104]. In this thesis, we

only focus on run-time sources of uncertainties; therefore, uncertainties that originate from the requirements and design phase of the systems are out of the scope of this work. We classify the run-time uncertainties as: 1) internal—originating from the CPSs, and 2) external—originating from the unpredictability of the environment (i. e., the context) in which the CPSs operate. The internal run-time uncertainties that originate from the CPSs themselves are usually due to some technical limitations. For example, different sensor uncertainties: sensor ambiguity, imprecision, or even complete sensor failures [104], or other hardware and software failures in different components of the CPS that could potentially lead to an unwanted system's behaviour. Furthermore, the sensors of the CPSs have only a limited sensing range, meaning they can only make partial observations of the context in which they operate, which introduces another dimension of internal run-time uncertainties. On the other side, the external uncertainties stem from the inherently changing and often unanticipated contexts in which the systems operate, and can manifest in various forms depending on the system under consideration. A common approach to deal with run-time changes and uncertainties that could not be fully anticipated during the system's design is to make the CPSs self-adaptive.

1.1.2 Self-Adaptation as an Emerging Property of Modern Systems

In the literature, self-adaptation is widely considered an effective approach that enables any software or CPS to autonomously deal with various dynamics emerging from the context or the system itself. Namely, self-adaptation aims in supporting the engineering of the systems that will have the ability to autonomously handle changes and uncertainties during the run-time or operational time [95, 73, 84]. The core idea behind self-adaptivity is the general systems' ability to change their behaviour, parameters or structure, as a response to internal and external changes and uncertain execution conditions without human intervention in order to continue meeting their system goals.

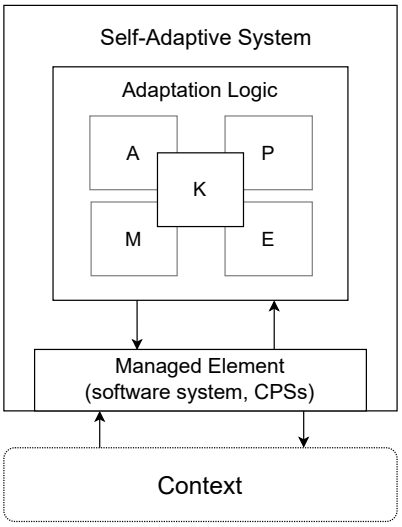


Figure 1.1: *Conceptual model of a self-adaptive system [118, 66].*

Although the terms system adaptation and self-adaptive systems have not been precisely defined yet [118, 116], over the years, various approaches for engineering self-adaptive systems have been proposed in the literature. The two most prominent ones are: control-based [41, 6] and architecture-based [66, 28, 43, 119] self-adaptation. In this work we mainly focus on the latter. In architecture-based self-adaptation, it is broadly accepted that a self-adaptive system on an abstracted level is comprised of a *managed element* and an *adaptation logic*, as depicted in Figure 1.1. In previous works, the adaptation logic has also been referred to as managing system [118], or as autonomic manager [66]. The managed element can be either a traditional software system or a CPS. It is the entity that obtains self-adaptation capabilities, given by the adaptation logic. The adaptation logic is a part of the self-adaptive system that gives the ability to the managed

element to self-adapt. A common approach to realize the adaptation logic is through the MAPE-K (Monitor, Analyze, Plan, Execute) feedback loop, with shared Knowledge among all the components of the loop [66]. Furthermore, as explained previously, every system operates in a context.

1.1.3 Challenges while Engineering Self-Adaptive Systems

Besides computer science, there are other fields and disciplines that have been considering the notions of adaptation, for example, biology and evolutionary sciences [16, 127], climate change and environmental sciences [89, 42], as well as film, cinematography and media studies [56, 79]. Concretely in the field of software engineering, there exist many prior efforts to define self-adaptive systems informally [111, 120, 73, 32, 117], as well as formally [20, 120, 22, 23]. However, despite the notable advancements in the research on system (self-)adaptation in the last two decades and the domain's active community, none of the previously proposed definitions has been broadly accepted, and the shared understanding of the core terminology is still imprecise. According to Weyns, self-adaptive systems have not been defined yet [116], and the lack of precise and broadly accepted definitions is possibly the biggest challenge in the field of engineering self-adaptive systems [118, 119].

Separate prior works in the literature [20, 82, 116] have independently observed that the terms adaptation, and respectively, self-adaptive systems are primarily used intuitively without deeper understanding and explanations of their precise meaning. The intuitive interpretation of the notion of adaptation, which primarily relies on the common language, is ambiguous, resulting in underspecified usage of these terms. Although in some instances such usage might suffice, this is not the case in engineering and science, where if and when a system behaves adaptively cannot be answered by means of intuition, and more rigorous definition is necessary [82]. *Understanding and defining system adaptation is the first step towards defining self-adaptive system.*

The intuitive usage of the terminology can be further observed in various other works that have taken a different stance towards system (self-)adaptation, frequently using the terms adapting and changing, even dynamic and flexible, interchangeably. For example, in a recent work [25] the authors claim that every cyber-physical system (CPS) is self-adaptive. Even more extreme, one might argue that an `if-then-else` statement, where the condition is related to the input values, already incorporates the notion of adaptation. As another example, we consider an informal definition of adaptivity by Broy [17]: "The core concept behind adaptivity is the general ability to change a system's observable behaviour, structure, or realization basically without users' interaction." However, one might have an intuitive question: does not (almost) every system change its behaviour based on the received inputs? For instance, robots moving in the room autonomously and discovering new tasks that are later attained by the robots already change the observable behaviour of the systems. Thus, we identify a significant challenge towards defining self-adaptive systems: *the importance of understanding and differentiating when a system functions and when it adapts.*

The lack of definitions on *what* are self-adaptive systems, has different software engineering consequences and implications, for instance, *how* to build and evaluate these systems.

Therefore, understanding, characterizing and defining self-adaptive systems, is the first step towards specifying, modelling and implementing these systems. This should also enable a more critical and systematic comprehension of the existing works in this domain. Finally, having a more precise definition sets the foundation on *how* to evaluate and compare these systems in the future, i. e., it is pivotal for the rest of the engineering phases, e. g., testing and verification.

1.2 The Goal of this Thesis

The overarching research objective of this thesis is to broaden the understanding and provide a unifying formal foundation of the notions of system adaptation and self-adaptive systems in software and systems engineering. So far in the literature, we have seen that these terms have been used liberally and mostly intuitively, without deeper insights and a clear distinction between a system operation (i. e., system function) and a system adaptation. For improved clarity and coherence—potentially yielding to increased prominence of this domain, we consider as essential to gain a better comprehension of the semantics of the terminology. Ideally, being able to better frame adaptation as a system property, and identify and specify characteristics of self-adaptive systems should enable us to have more constructive discussions on the existing and current works, as well as identify future challenges and directions.

In a nutshell, **the first goal of this thesis is to establish definitions and formulate *what* is system adaptation, which sets the foundation for defining self-adaptive systems.** This helps in identifying the characteristics and the properties of these systems, and it serves as the first step towards discussing *how* to engineer them. Our aim with the first goal is not to disregard all the previous work in this research domain but to complement the existing works with improved semantics, which in result, will yield to improved clarity and comprehension of the terminology. As a **second goal of this thesis, we focus on *how* to engineer a specific class of self-adaptive CPSs.** Concretely, within the frame of the proposed formal definitions, we further propose architectural, methodological, and technical solutions for engineering self-adaptive CPSs, which are inherently decentralized and operate in changing, uncertain, and partially observable contexts.

1.3 Problems Statements and Research Gaps

From the thesis' goals described in the previous section, we derived a few theoretical and practical research problems, classified into two major parts: **Part 1—Theoretical foundations for system adaptation and self-adaptive systems** and **Part 2—On engineering self-adaptive CPSs in dynamic context**. For each problem, we identify a corresponding gap that the existing body of literature has not considered, to which this thesis contributes. The problems and the respective gaps this thesis addresses are summarized in this section. Additionally, at the end of Part 1, we also provide a brief summary of some of the theoretical findings that result in different engineering implications for self-adaptive systems and serve as a foundation for identifying the problems in Part 2.

Part 1: Theoretical Foundations for System Adaptation and Self-Adaptive Systems

Problem 1: *Lack of definitions of system adaptation and self-adaptive systems in software and systems engineering.* Despite the past efforts, the notion of system adaptation and self-adaptive systems have not been precisely defined in the literature. Understanding and defining system adaptation is the core pillar towards defining self-adaptive systems. Although there might be some intuitive understandings, including some informal definitions of these notions, the terminology is still ambiguous, and therefore, understandable, and universally practicable definitions are missing. If we do not know *what* self-adaptive systems precisely are, we cannot argue how to engineer them. This need increases further with the growing complexity of modern and dynamic systems. Moreover, the lack of commonly accepted definitions makes it difficult for the systems and different solutions in this research domain to be compared and discussed, since the core terminology is used with different semantics.

Gap 1: The literature in software engineering lacks a precise, comprehensive and broadly accepted formal definition of system adaptation and self-adaptive systems.

To the best of our knowledge, there are only a few works in the literature that provide a formalized world of concepts for self-adaptive systems [20, 120, 22, 23, 7, 129]. Among them, two of the previous works by Broy et al. [20] and Bruni et al. [22] have tried to identify how adaptive systems differ from the ordinary, non-adaptive systems. However, none of the past efforts has studied the exact difference between nominal system functioning and system adapting, which is the essential starting point for a further discussion and definition. To this contributes differentiating the system goals in self-adaptive systems in two categories: business and adaptation goals. Concretely, the core motivation for system adaptation is that self-adaptive system is able to continue meeting their functional specifications (i. e., business goals, concerns of the managed system), while maintaining or even improving adaptation goals, which are one or more quality objectives (concerns of the adaptation logic). Moreover, the existing works in the literature do not discuss 1) the necessary framing in order to discuss system adaptation, 2) the shared characteristics of self-adaptive systems, nor 3) identify the criteria and the prerequisites for a system to be self-adaptive. The analysis of the related work that identified this gap is presented in Section 8.1.

Short summary of some of the theoretical findings

In the following, we provide a brief summary of some of the theoretical findings on system adaptation and self-adaptive systems that are further elaborated, discussed, and deduced in Chapter 5 and Appendix A. These theoretical findings lead to increased conceptual clarity and characterization of (self-)adaptive systems and have various engineering implications for developing these systems. We summarize these findings here since they serve as a basis for identifying the upcoming research and engineering problems in Part 2 of this thesis.

On system adaptation:

- Adaptation is a property of an individual system function. However, an individual function does not mean a simple function and the granularity of the function matters. *The complexity for adaptation is proportional to the complexity of the system function that adapts.*
- The system (the managed element) always adapts to certain adaptation goals (i. e., one or more quality objectives). The same system (i. e., the system function) might be adaptive according to one adaptation goal, and maladaptive according to another.
- The business goals are related to the functional requirements of the managed element or the system (i. e., the system function); whereas, the adaptation goals are one or more quality objectives, associated with specific non-functional requirements and are concerns of the adaptation logic.
- The system (i. e., the system function) can only adapt to certain context (or system) situations. No system universally behaves adaptively in every condition. *The complexity for adaptation is proportional to the complexity of the context and the system that gains adaptation capabilities.*

On self-adaptive systems:

- Various uncertainties and changes can lead to situations in which the adaptation goals (i. e., the system adaptation) are not fulfilled, which presents a trigger for the system to self-adapt.
- The system self-adapts with intention to eventually achieve system adaptation.
- We differentiate between passive and active self-adaptation. In the passive self-adaptation the system (i. e., the managed element) is enriched with an adaptation logic built according to the MAPE-K. In the active self-adaptation, to which we refer to as true self-adaptation, the adaptation logic is also extended with a functionality that enables the system to evaluate the fulfilment of the adaptation goals (i. e., the system adaptation), based on which the need for self-adaptation is triggered and different actions towards adaptation are selected.
- The same system (i. e., the system function) adapts differently according to different adaptation logic..

On the adaptation logic:

- The adaptation logic is constructed based on the concrete system function that gains adaptation capabilities, the adaptation goals and the uncertainties according to which the managed element (i. e., the system) is considered to be adaptive.

- Assuming that the adaptation logic is built according to the MAPE-K. Since the implementation of all the other MAPE phases can be distributed in the managed element [123], the very minimal requirement for the system to be self-adaptive is the existence of an additional knowledge in the adaptation logic (see Figure 1.1) created correspondingly to the adaptation goals and the relevant aspects for the adaptation (tackled in Problem 2).
- The knowledge should *reflects* the relevant aspects for the adaptation from the state, the model, or the behaviour of: 1) the system (i. e., the managed element), 2) the context, or both.
- The knowledge in the adaptation logic of the self-adaptive system can be hard-coded during design time. However, if the system and its context are dynamic and change during run-time (e. g., in the case of CPSs), then in order for the knowledge in the adaptation logic to reflects their actual state, behaviour or structure, it also needs to be modified during run-time (tackled in Problem 2 and 3).
- To update the knowledge during run-time, there is a need for reasoning on the relevant aspects regarding the specific adaptation from what each system (i. e., CPS) observes from its dynamic, uncertain and partially observable context (tackled in Problem 2 and 3).
- The knowledge and the other components of the adaptation logic, need to capture the uncertainties introduced during run-time (tackled in Problem 3).

On (system and context) changes and uncertainties:

- A system cannot adapt to unknown unknowns. While discussing self-adaptive systems, it is essential to identify what aspects that are relevant for the adaptation are changing (i. e., what is and the degree of unknown or unanticipated), according to which we want to adapt considering the specific adaptation goals (tackled in Problem 3).
- Similarly, the uncertainties for which the concrete adaptation accounts need to be identified and characterized, and their type and range need to be defined (tackled in Problem 3).

The newly gained, more profound understanding of system adaptation and self-adaptive systems helps us characterize these systems better and determine the components that need to be inherently part of a self-adaptive system, as well as what kind of information and functionality those components provide. As previously explained, in this thesis, we are concretely interested in engineering autonomous and decentralized single or multi-agent self-adaptive CPSs (MA-SACPSs), which operate in changing, uncertain, and partially observable contexts. The engineering of this class of self-adaptive CPSs is what we tackle in Part 2 of this thesis.

Part 2: On Engineering Self-Adaptive CPSs

Problem 2: *Lack of architectures for engineering (MA-)SACPSs operating in changing, uncertain, and partially observable contexts.* As a consequence of the scarcity of definitions of system adaptation and self-adaptive systems, and the lack of clear characterization of these systems, there is a deficiency of architectures and design patterns that can serve as a blueprint for engineering self-adaptive systems. Namely, establishing definitions and understanding *what* are self-adaptive systems, is the first step towards specifying, modelling and designing these systems. However, to define what are self-adaptive systems, we, first and foremost, need to define system adaptation. Also having definitions would mostly remain declarative and descriptive, as they do not provide constructive insights on *how* self-adaptive systems are designed and build.

The MAPE-K conceptual model is the only broadly accepted model in the literature for engineering self-adaptive systems. Although MAPE-K gives some intuition behind the engineering of self-adaptive systems, primarily by the separation of concerns between the managed element and the adaptation logic, it still lacks a more precise semantics of these two components. Hence, there are a few problems emerging from the usage of the MAPE-K as a reference model for engineering self-adaptive systems. *First,* it can often be interpreted that the managed element, which is just any system, already contains elements of monitoring, analysis, planning and execution; as a result, blurring lines between the managed element and the adaptation logic. This separation becomes additionally blurry with the introduction of different MAPE-based patterns for self-adaptive systems [123, 101]. *Second,* the conceptual MAPE-K model has a very high level of abstraction and is not particularly helpful in designing and the implementation of the actual system. Concretely, there is a big disconnect between the MAPE-K (see Fig. 1.1) and potential technical implementations of a self-adaptive system. *Third,* the conceptual MAPE-K model does not provide any concrete insights on how self-adaptive systems (engineered according to MAPE-K) differ from the ordinary systems that are non-adaptive, *Finally,* it is not clear how self-adaptive systems engineered according to the MAPE-K differ from the other self-* systems (e. g., self-awareness [81, 72], self-healing [100, 49]), since MAPE-K is used for engineering all the self-* systems in general. As a result, a need emerges for an architecture for engineering self-adaptive systems at some intermediate level of abstraction which conforms to the formal definitions and the more precise characterization of self-adaptive systems. As previously explained, in this thesis, we are concretely interested in engineering autonomous and decentralized (MA-)SACPSs, which operate in changing, uncertain, and partially observable contexts.

Gap 2: The existing works in the literature do not provide concrete architectures, frameworks nor methodologies for engineering self-adaptive systems, particularly decentralized and autonomous (MA-)SACPSs that operate in changing and uncertain contexts that are only partially observable by the CPSs.

Until now, in the literature, there are several proposed design patterns for self-adaptive systems [123, 101]; however, these works 1) mainly focus on different combinations in the decentralization of the four phases of the MAPE and 2) they completely exclude the

explicit consideration of the knowledge component in their patterns. Although different MAPE-based patterns are more informative regarding the system's design, inherently they have the same limitations as the MAPE-K closed feedback loop itself: their high level of abstraction and low level of details, which does not provide any characterization of how a system built upon the MAPE loop differentiates from the *ordinary*, non-adaptive systems. The complete analysis of the related work that identified this gap is presented in Section 8.2. In the following we summarize relevant information regarding the architectural solution, necessary to identify and introduce the third and the final problem of this thesis.

A potential architectural solution should consider all the identified components of a self-adaptive system that were previously discussed in the theoretical findings. However, since it is a general solution, it shall not prescribe any use case or system-specific properties, e. g.:

- the relevant aspects from the state and the behaviour of the CPSs and the context for the concrete adaptation,
- how the knowledge in the adaptation logic is modelled according to those relevant aspects,
- the relevant uncertainties for the specific adaptation, and
- how to reason under uncertainties in order to update the knowledge in the adaptation logic during run-time.

All these aspects need to be conveniently considered by the engineers of the self-adaptive systems, but they can only be answered within the frame of the specific use case or system under consideration.

Problem 3: *Lack of solutions for knowledge and uncertainty representation in the adaptation logic, and reasoning under uncertainty to update the knowledge during run-time.* Knowledge modelling and representation in the adaptation logic, and run-time reasoning based on which that knowledge is updated are important for building self-adaptive systems, especially self-adaptive CPS that are dynamic themselves and operate in dynamic, changing and uncertain context that is often only partially observable by the CPSs. In dynamic systems and contexts, in order for the "best" adaptation to be feasible, the knowledge in the adaptation logic should be continuously updated to reflect the run-time state of the context (and the system), relevant for the concrete adaptation. As a result, having an encoded context model in the knowledge in the adaptation logic during the design of a dynamic self-adaptive CPS does not suffice the adaptation during the system's run-time. In response, the need emerges for an efficient representation of the knowledge, which ultimately enables real-time updating of the knowledge. Since the monitoring technology of the CPSs, e. g., the sensors, introduce various types of (internal) run-time uncertainties besides the (external) uncertainties from the context in which the systems operate, the knowledge cannot be updated directly upon the observations made by the CPSs. Instead, there is the need for reasoning under uncertainties before the knowledge is updated, which will enable accurate representation of the actual state of the context or the system (i. e.,

the managed element) during run-time. The updated knowledge in the adaptation logic is later used in different phases of the MAPE loop to further analyze and plan the next adaptation actions. The run-time reasoning can be considered as a run-time uncertainties mitigation strategy in self-adaptive systems. Additional details on the uncertainties in dynamic systems are given in Section 2.1.

Gap 3: There is a scarcity of approaches proposed in the literature that allow domain- and system-independent modelling of the knowledge in the adaptation logic and run-time reasoning, based on which the knowledge is continuously updated to accurately reflect the current state of the relevant aspects from the context and the system for the concrete adaptation. Although knowledge representation and reasoning are essential for building self-adaptive CPSs, especially MA-SACPSs, there is a scarcity of approaches for modelling the knowledge in the adaptation logic that allow capturing uncertain observations from single or multiple, decentralized CPSs, to effectively reason and aggregate the observations, and eventually update the knowledge. The observations that the CPSs make using their sensors are uncertain, i. e., faulty, inaccurate, and in response, potentially conflicting. Before knowledge is updated, those uncertain observations need to be aggregated. However, so far in the literature, knowledge representation in the adaptation logic has been treated as a problem- and system-specific task [44]. Also, to our knowledge, there is no other work that contributes toward run-time observations aggregations and reasoning under uncertainties based on which the knowledge in the adaptation logic is updated in order to reflect the run-time state of the dynamic context of the self-adaptive CPSs. The analysis of the related work on mitigating uncertainties in self-adaptive systems is presented in Section 8.3.

1.4 Solution

This thesis contributes towards refining the existing engineering processes for self-adaptive systems by proposing different theoretical, architectural, methodological and technical solutions for the problems identified in the previous section, summarized and shown in Figure 1.2. The figure also depicts the generality of the proposed solutions. Furthermore, the proposed solutions as part of this thesis support the engineering of both single-agent self-adaptive CPSs and multi-agent self-adaptive CPSs (MA-SACPSs).

As part of this thesis, we present a three-fold solution to address **Problem 1** (*Lack of definitions of system adaptation and self-adaptive systems in software and systems engineering*). In the first solution, we first propose a taxonomy of autonomous systems and formally specify different levels of autonomy. During our efforts to define self-adaptive systems, we realized that gaining a more profound understanding of system autonomy is necessary before scoping and discussing self-adaptive systems. Our taxonomy of autonomous systems served as a backbone for differentiating different types of self-adaptive systems in the later solutions of this thesis.

In our second solution, we aimed to systematically investigate how self-adaptive systems are defined across the literature. Although the objective of this thesis was to define self-adaptive systems, during the course of this research, it became clear that in order to

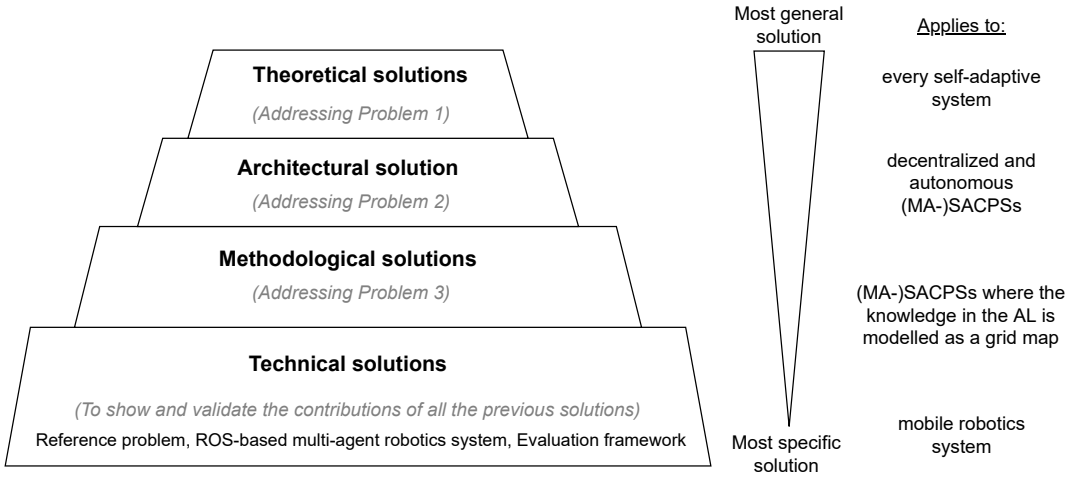


Figure 1.2: *Trapezoid of the solutions proposed in this thesis and the generality of different solutions.*

define self-adaptive systems, we first need to define the concept of *system adaptation* in software and systems engineering. In response, as part of this study, we investigated if the existing efforts also formally define system adaptation as part of their contributions and if they consider the characteristics of self-adaptive systems that we summarized from two principles on self-adaptive systems proposed by Weyns in a recent work [116]. In our study, we also discussed the limitations and shortcomings of the related efforts and elicited the requirements for a holistic and formal definition of self-adaptive systems.

In the third solution to the first problem, we formally define system adaptation and identify how the system function that is considered adaptive, the context, and the separation between business and adaptation goals (i. e., some quality objectives) play a central role in defining system adaptation, and respectively, self-adaptive systems. Our formal contribution increases the semantics of the MAPE-K conceptual model, which has been proposed in the literature as a reference model for engineering not only self-adaptive but self-* systems in general. In addition to the formal definition of system adaptation, we propose a framing that is necessary to be considered in order to answer if a system is adaptive. The definition of system adaptation and the proposed framing set the basis for our formal definitions of self-adaptive systems.

In our formalisms, we adopt an approach to define system adaptation using the function considered to behave adaptively. Based on our definition of system adaptation, we propose a metric for measuring system adaptation, which we refer to as Quality Function. We also identify how context, the differentiation between adaptation and business goals, and the consideration of knowledge in the adaptation logic play a central role in defining passive self-adaptation. In active self-adaptation, the self-adaptive system additionally evaluates itself (using the Quality Function), based on which it can detect when the system adaptation is not fulfilled, which presents a trigger for self-adaptation. In response, various actions towards adaptation could be chosen with the intention for system adaptation to be potentially achieved again. All the theoretical findings, formal solutions, and contributions

proposed in this thesis present the foundation for specifying, designing, and evaluating self-adaptive systems, and they also serve as a foundation for comparing the existing works.

To address **Problem 2** (*Lack of architectures for engineering (MA-)SACPSs operating in changing, uncertain, and partially observable contexts*), we embed our formal definitions of system adaptation and self-adaptive systems into an applicable logical architecture for engineering self-adaptive CPSs that narrows the gap between the formal foundations and potential technical implementations of a class of systems. Concretely, the proposed logical architecture can serve as guidance or a blueprint for engineering autonomous and decentralized (MA-)SACPSs that operate in changing and uncertain contextual conditions. In our logical architecture, not only that the managed elements (i. e., the CPSs) adapt, but the adaptation logic (precisely the knowledge in the adaptation logic) changes itself during run-time to accommodate changes and uncertainties that could not be anticipated during the design of the self-adaptive system. In the logical architecture, we put a special emphasis on considering the knowledge in the adaptation logic, and the procedure established in our solution can also be seen as methodological guidance for updating the knowledge during run-time in order to reflect the actual run-time state and behavior of the systems and the context. It is important to emphasize that in our logical architecture, we do not prescribe how to represent the knowledge in the adaptation logic, nor the type and the range of the uncertainties, including methods to reason upon them.

In a nutshell, our proposed formal definitions of adaptation can be considered orthogonal to architectures on different levels of abstraction: 1) the MAPE-K conceptual model with the highest level of abstraction and the lowest level of details, 2) the proposed logical architecture with an intermediate level of abstraction and intermediate level of details, and 3) a potential technical architecture of an implementation—instantiated from our logical architecture–with the lowest level of abstraction and the highest level of details.

To address **Problem 3** (*Lack of solutions for knowledge and uncertainty representation in the adaptation logic, and reasoning under uncertainty to update the knowledge during run-time*), we propose a Subjective Logic-based methodological approach for knowledge and uncertainty representation, which enables reasoning under uncertainties of the decentralized monitoring by the CPSs. Subjective Logic (SL) [60, 61] is an enriched probabilistic logic-based framework for artificial reasoning, based on the Dempster-Shafer Theory (DST) [34, 109] of evidence. (MA-)SACPSs inherently produce uncertain, partial, faulty and potentially conflicting observations from the relevant aspects of the context for the concrete adaptation. Therefore, the proposed methodology supports run-time reasoning and observations aggregation based on which the knowledge in the adaptation logic is updated. This enables the knowledge to have the "best" representation of the run-time state of the dynamic and uncertain context and serves as a basis for analysing and planning the next adaptation actions. By "best" representation, we refer to the most complete, consolidated and accurate representation of the relevant aspect from the context for the concrete adaptation, in which different uncertainties and potential conflicts that stem from the uncertainties are resolved. The uncertainties could introduce various imparities between the actual state of the context and the modelled context in the knowledge in the adaptation logic. The updated knowledge is later utilised for planning the next adaptation actions.

Finally, although the problems presented in Section 1.3 are general, the solutions, particularly the solutions to Problem 3, are only possible when considered in a specific domain of applications. For that reason, to show, to evaluate and validate the contributions of all the previous solutions, we provide a few other technical artifacts as part of this thesis, including 1) a reference problem from the robotics domain, 2) an implementation of the ROS-based robotics system for experimentation with i) the implementation of the (MA-)SACPSs, ii) different types of internal and external uncertainties and iii) the dynamics of context and, 3) an evaluation framework to evaluate the system adaptation according to our definitions and formalisms.

1.5 Contribution

The thesis makes the following contributions w.r.t. the gaps in the literature identified in Section 1.3. Concretely, to close *Gap 1*, we make the following contributions:

C1 A taxonomy of autonomous systems. To better frame the need for self-adaptive systems, we propose a taxonomy that supports formal specifications of different levels of autonomous systems. We discuss how a certain level of autonomy is a precondition for self-adaptation and also show that self-adaptation is necessary for achieving a higher level of autonomy since it enables the system to deal with various changes and uncertainties that were not (fully anticipated) during the design of the system.

C2 Systematic literature review on defining self-adaptive systems. In recent years, self-adaptation has received growing attention in software and systems engineering; however, a precise understanding of these systems is still lacking. There have been a few attempts to define self-adaptive systems over the years, but there is still no consensus on the definition of these systems. This is probably because the existing efforts did not aim to understand and define system adaptation prior to defining self-adaptive systems. Towards narrowing **gap 1**, we have systematically reviewed how system self-adaptive systems have been previously formally defined and characterized in the literature. In this work, we also analyze the limitations and shortcomings of the existing formal definitions.

C3 Formal definition of system adaptation and self-adaptive systems. To close **Gap 1**, we propose 1) a formal definition of the notion of system adaptation, 2) a framing for engineering (self-)adaptive systems, and 3) a theoretical framework in which we identify and define of two types of self-adaptive systems. We additionally specify the process of self-adaptation, elicit the minimum requirements for self-adaptive systems and discuss various architectural implications from our theoretical contributions. Concretely, in our formalisms, we adopt an approach to define system adaptation using the function considered to behave adaptively. In our formal definitions of self-adaptive systems, we differentiate between passive and active self-adaptation based on the ability of the system to evaluate itself and indicate when the system adaptation is not fulfilled.

To tackle *Gap 2*, the following contribution is proposed:

C4 A logical architecture for engineering self-adaptive CPSs in dynamic context. To close **Gap 2**, we propose a logical architecture for engineering (MA-)SACPSs operating in a dynamic, uncertain, and partially observable context that builds upon our formalisms for system adaptation and self-adaptive systems, previously proposed to close Gap 1. The proposed architecture narrows the gap between the MAPE-K as a conceptual architecture and the formal foundations, and concrete technical implementation. Furthermore, the logical architecture can be considered as procedural guidance for dynamic knowledge improvement in the adaptation logic.

To fill *Gap 3* we make the following contribution:

C5 A methodology for knowledge representation and run-time reasoning under uncertainties in self-adaptive CPSs. To close **Gap 3**, we propose a methodology for knowledge modelling, aggregation and reasoning in (MA-)SACPSs that is domain- and system-independent and can deal with reasoning on uncertain, partial and conflicting observations. Concretely, our approach uses Subjective Logic (SL) to update the knowledge in the adaptation logic at run-time by aggregating partial observations of the relevant aspects of the context for the concrete adaptation made by each CPSs in (MA-)SACPSs.

And the last contribution of this thesis is the following:

C6 A model problem, a ROS-based robotics system and an evaluation framework for evaluating system adaptation and experimentation with self-adaptive CPSs and the dynamics of the context. Finally, we propose a model problem from the domain of robotics, a ROS-based simulated robotics system and an evaluation framework to evaluate system adaptation according to our definitions and formalisms. These three artifacts provide a foundation and enable researchers and practitioners in this domain to build, experiment, evaluate and compare self-adaptive systems and different aspects of system adaptation.

Figure 1.3 highlights the contributions of this thesis. As part of this publication-based doctoral thesis, these contributions have previously appeared in the following peer-reviewed publications published in proceedings of international conferences and journals:

P1 S. Kugele, **A. Petrovska**, I. Gerostathopoulos, "Towards A Taxonomy of Autonomous Systems," 15th European Conference on Software Architecture (ECSA), 2021.

P2 **A. Petrovska**, G. Erjiage, S. Kugele, "Defining Self-Adaptive Systems: A Systematic Literature Review," IEEE/ACM 18th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2023. *Under review.*

P3 **A. Petrovska**, S. Kugele, T. Hutzelmann, T. Beffart, S. Bergemann, A.Pretschner, "Defining Adaptivity and Logical Architecture for Engineering (Smart) Self-Adaptive Cyber-Physical Systems," Information and Software Technology Journal, 2022

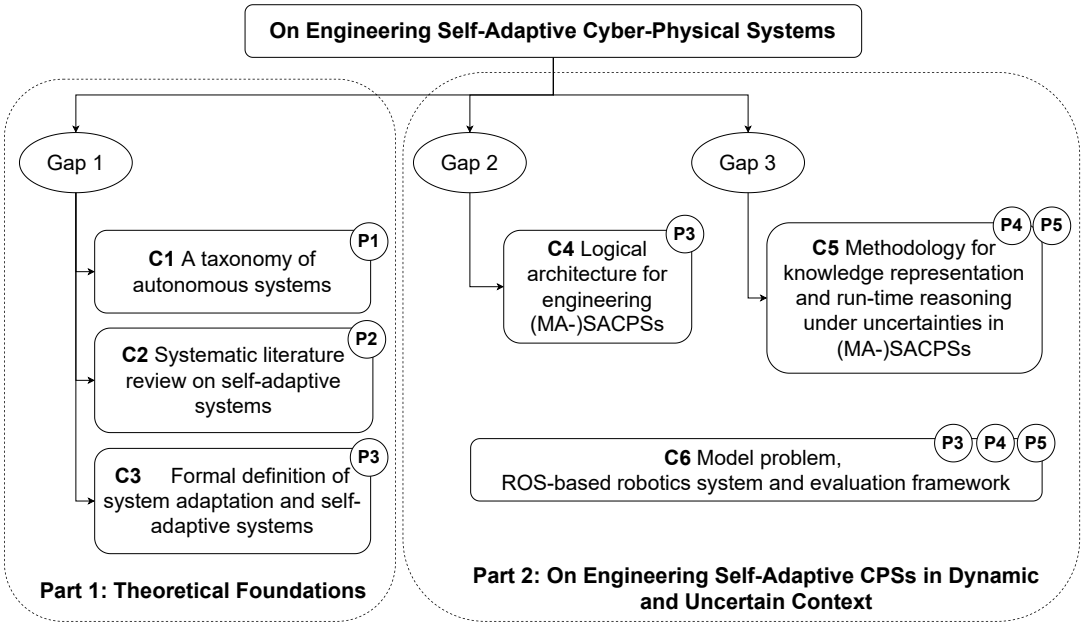


Figure 1.3: *Mapping of Gaps, Contributions and Publications to the structure of this thesis.*

P4 **A. Petrovska**, S. Quijano, I. Gerostathopoulos, A. Pretschner, "Knowledge Aggregation with Subjective Logic in Multi-Agent Self-Adaptive Cyber-Physical Systems," IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2020.

P5 **A. Petrovska**, M. Neuss, I. Gerostathopoulos, A. Pretschner, "Run-time Reasoning from Uncertain Observations with Subjective Logic in Multi-Agent Self-Adaptive Cyber-Physical Systems," IEEE/ACM 16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2021.

In addition to the previously enumerated papers, the author of this thesis have co-authored a few additional peer-reviewed publications. The papers tackle relevant problems, related to the topic of this thesis; however, they are excluded from this thesis.

1. C. Hildebrandt, T. Bandyszak, **A. Petrovska**, N. Laxman, E. Cioroaica, "EURECA: Epistemic Uncertainty Classification Scheme for Runtime Information Exchange in Collaborative System Groups," International Journal on Software-Intensive Cyber-Physical Systems (SICS), 2018.
2. **A. Petrovska**, F. Grigoleit, "Towards Context Modeling for Dynamic Collaborative Embedded Systems in Open Context," 10th International Workshop Modelling and Reasoning in Context (MRC), 2019.
3. **A. Petrovska**, A. Pretschner, "Learning Approach for Smart Self-Adaptive Cyber-Physical Systems," IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), 2019.
4. **A. Petrovska**, M. Neuss, S. Bergemann, Martin Büchner, M. Ansab Shohab, "Smart Self-Adaptive Cyber-Physical Systems: How can Exploration and Learning Improve

Performance in a Partially Observable Multi-Agent Context?," 13th International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE), 2021.

5. **A. Petrovska**, J. Weick, "Bayesian Optimization-Based Analysis and Planning Approach for Self-Adaptive Cyber-Physical Systems," IEEE 2nd International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), 2021
6. **A. Petrovska**, "Self-Awareness as a Prerequisite for Self-Adaptivity in Computing Systems," 6th Workshop on Self-Aware Computing (SeAC), 2021
7. **A. Petrovska**, J. Weick, "Realization of Adaptive System Transitions in Self-Adaptive Autonomous Robots," The 37th ACM/SIGAPP Symposium On Applied Computing (SAC), 2022

1.6 Structure

In Chapter 2, we provide the overview of the background and the required formal foundations of the Subjective Logic. The solutions and the contributions to the presented problems and gaps are described in Chapters 3 to 7. Chapter 3 proposes the taxonomy and formalization of different levels of autonomous systems. In Chapter 4, we present the systematic literature review, in which we investigate the existing formal definitions of self-adaptive systems in the literature and analyze and summarize the limitations of the formal definitions. Some findings from the proposed taxonomy of autonomous systems and the limitations of the existing formal definitions of self-adaptive systems serve as a basis for the rest of our theoretical contributions. In Chapter 5, we provide the framing and the formal definition of system adaptation, and discuss the preliminary characterization and minimum requirements for self-adaptive systems. Also, as part of this chapter, we propose the logical architecture for engineering self-adaptive CPSs that operate in dynamic and uncertain contexts. As discussed prior in the introduction, parts of the logical architecture can be considered as procedural guidance for changing the knowledge in the adaptation logic during run-time; however, as part of the logical architecture, we do not prescribe how the knowledge can be exactly modeled. In response, in Chapters 6 and 7, we propose the methodology for knowledge representation and reasoning under uncertainty. In Chapter 8, we summarize the related work. In Chapter 9, we conclude this thesis by discussing the results and summarizing the contributions of this thesis and providing an outlook to the future work. In Appendix A, we complement the theory from Chapter 5 by proposing a theoretical framework for engineering self-adaptive systems. In this framework, we make the following contributions: (1) formally define passive and active self-adaptive systems, (2) we extend the minimum requirements for a system to be self-adaptive, and (3) we identify and specify the process of self-adaptation.

2 Background

This chapter presents the background on uncertainty classification and various theories for reasoning under uncertainties, including the formal foundations of Subjective Logic. As part of this thesis, we use Subjective Logic to represent and reason upon uncertainties during run-time, based on which the knowledge in the adaptation logic is updated. Parts of this chapter have previously appeared in publications [97, 98], co-authored by the author of this thesis.

2.1 Uncertainty Classification and Run-Time Uncertainty Taxonomy

The most general classification of uncertainties in the literature distinguishes between aleatory and epistemic uncertainties. Aleatory uncertainties (also known as aleatory variability) exist from the natural randomness of the processes [1], and the modeller does not foresee the possibility of reducing them [35]. Whereas uncertainties are characterized as epistemic if the modeller sees a possibility to reduce them by gathering more data or by refining models [35]. Usually, they exist due to limited data and knowledge.

In the domain of self-adaptive systems, Ramirez et al. [104] have previously proposed a taxonomy of uncertainty for dynamically adaptive systems. In dynamic and adaptive systems, the uncertainty originates in one of the following three phases: requirements, design and run-time phase. If the uncertainty is not resolved within the phase it originates, it can be propagated to the subsequent phases. Ultimately, any uncertainty that is not resolved in the requirements and design phase propagates at run-time. There is a great variety of sources of uncertainty in any of the three phases. Table 2.1 gives an overview of the most common sources of uncertainty, including their definition and existing mitigation strategies. In this thesis, we only consider uncertainty sources that occur in the run-time phase.

	Id	Term	Definition
Requirements	1	Missing Requirement	The specification is incomplete and does not cover all requirements.
	2	Ambiguous Requirement	The specification or evaluation criteria can be interpreted in different ways.
	3	Falsifiable Assumption	A possibly false statement used to support the validity of a requirement.
	4	Unsatisfiable Requirements	A requirement that cannot be satisfied by the dynamically adaptive systems.

	5	Requirements Interactions	Two or more requirements that inadvertently interfere with each other.
	6	Doubt	A concern about a statement or requirement.
	7	Claim	A subjective rationale that forms the basis of a decision.
	8	Changing Requirements	Requirements that do not reflect the needs and constraints of the system.
Design	9	Unexplored Alternatives	When not all different design options are considered.
	10	Untraceable Design	Irrelevant design decisions from the perspective of requirements.
	11	Risk	An exposure to danger or loss.
	12	Misinformed Tradeoff Analysis	Design decisions based on misguided and subjective preferences.
	13	Inadequate Design	Requirements cannot be fully satisfied or include latent behaviours.
	14	Unverified Design	Lack of proof that shows a design satisfies its requirements.
	15	Inadequate Implementation	An implementation that contains errors or faults.
	16	Latent Behaviour	An unknown behavior that should be disallowed.
Run-time	17	Effector	An adaptation that alters the execution environment in unanticipated ways.
	18	**Sensor Failure**	When a sensor cannot measure or report the value of a property.
	19	**Sensor Noise**	Random and persistent disturbances that reduce the clarity of a signal.
	20	**Imprecision**	A lack of repeatability in a given measurement.
	21	**Inaccuracy**	A divergence between a measured value and its real value.
	22	**Unpredictable Environment**	Events and conditions in the environment that cannot be anticipated.
	23	**Ambiguity**	A lack of numerical precision and accuracy in a measurement.
	24	**Non-Specificity**	A property whose value is only known within a certain range of values.
	25	**Inconsistency**	Two or more values of the same property that disagree with each other.
	26	**Incomplete Information**	A missing or unknown dimension of data.

Table 2.1: *An overview of the various sources of uncertainty and their definitions, modified from Ramirez et al. [104].*

Besides the classification of the different sources of uncertainties (see Table 2.1), the authors in [104] also propose a taxonomy of the uncertainties in each of the three phases. The proposed taxonomies provide insight into the root causes and the potential effects of certain types of uncertainty. In Fig. 2.1 it can be seen that all the uncertainties at run-time origin either from the effectors, sensor failure, noise and imprecision, and unpredictable environment. As part of this thesis we exclude from consideration the uncertainties from the effectors, and we focus only on different sensor uncertainties and unpredictable environment. All the run-time uncertainties that we focus on in this dissertation are shown in bold in Tab. 2.1.

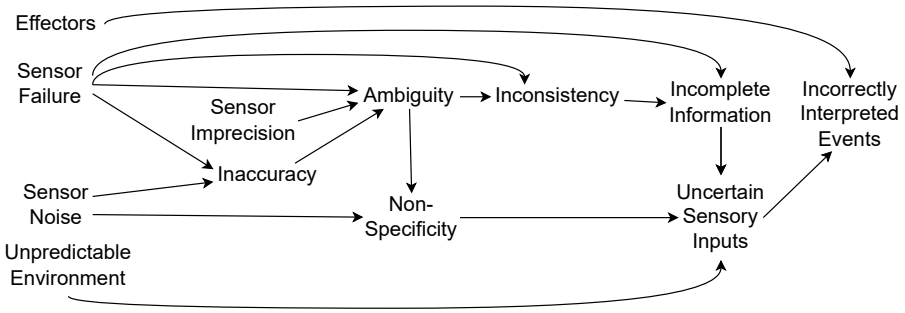


Figure 2.1: *The taxonomy of run-time uncertainty, created by Ramirez et al. [104].*

As part of this thesis, we further classify the sources of run-time uncertainties, as 1) *internal*—originating from the system itself, e. g., the various uncertainties from sensors and the uncertainties from the effectors as shown in Tab. 2.1, and 2) *external*—originating from the environment or the context in which the systems operate e. g., unpredictable environment from Tab. 2.1. Usually, it is infeasible for the developers of the MA-SACPS to anticipate at the system's design all the possible states of the context that the system will encounter during its operation. Consequently, the unpredictability of the environment will ultimately impart some uncertainty onto the MA-SACPS through its monitoring architecture.

Additionally, during its operation, a system can only perceive and determine parts of the context through the inputs for which the system has only encoded explicit interfaces (e. g., sensors). However, there can happen that there are aspects of the context that the system cannot determine through its sensors but still have (external) effects on the system's state. Similarly, there might be some internal aspects originating from the system itself (e. g., failure of a hardware component or a software bug) that also affect the system's state in a similar way as the external, contextual effects. In sum, we identify two additional dimensions of uncertainties: 1) the system may not be equipped with the necessary technology, i. e., lacks a concrete type of input (e. g., in case of robotics system, no sensor capable of detecting road conditions (external effects), or no sensor capable of

detecting a failed hardware component inside the system (internal effects), and 2) the technology used may be limited in its capability, i. e., the input cannot receive values outside of a specific range. For example, the CPS cannot observe the complete context in which they operate due to the technical limitations of their sensors, e. g., limited sensor range to which we refer as partiality or partial observations as part of this thesis. These newly introduced two dimensions can be considered as extensions from *incomplete information* and *non-specificity* from Tab. 2.1.

2.2 Reasoning under Uncertainty

There are various approaches to ascertain information in random, uncertain and unpredictable conditions. However, all rely on a few theories that describe how to reason under uncertainty. Two of the most prominent theories are the Bayesian Probability and the Dempster-Shafer Theory, which are further discussed in Section 2.2.1 and Section 2.2.2 respectively.

2.2.1 Bayesian Probability

Bayesian Probability is an interpretation of the concept of probability that can be seen as an extension of propositional logic. In Bayesian statistics, probability is used as the fundamental measure of uncertainty; therefore, it allows for reasoning with propositions whose truth values are uncertain. Being an evidential probability, the prior probability of a proposition (i. e., the probability of the proposition being true prior to any evidence being accounted for) is assigned, and as new evidence becomes available and accounted for, the probability of the proposition is updated through a mechanism called *bayesian updating* [93]. Unlike a frequentist view of probability, in which the probability of a proposition represents the frequency of the event occurring, in Bayesian Probability the probability of a proposition represents a state of belief [45].

Reasoning with Bayesian Probability is realized in three steps: (1) represent all sources of uncertainty as statistical random variables, (2) determine and assign a prior probability distribution to the random variables and (3) as more evidence is made available, update the probability distributions by applying the Bayes' formula:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}, \tag{2.1}$$

where $P(A)$ represents the prior probability of the proposition A being true, and $P(A|B)$ is the conditional probability of A being true given B is true. As new evidence becomes available, the probability distributions describing the propositions are updated, and these updated probabilities are then used as priors for further calculations with new evidence.

While Bayesian Probability appears to be a reasonably simple method of extending propositional logic to handle uncertainty, one issue that arises is when one wants to carry out abductive inference and is mistakenly assumed that $P(A|B) = P(B|A)$. Therefore, when one wants to reason backwards from some observable evidence to the likely hypothesis, the conditional probabilities must first be inverted [63]. Subjective Logic (further explained

in Section 2.3) supports both deductive and abductive reasoning as separate operators, which helps to avoid this mistake.

2.2.2 Dempster-Shafer Theory

The Dempster-Shafer theory of evidence (DST) introduced by Dempster [34] and Shafer [109] is a convenient and flexible theoretical framework to represent uncertain data and provides a method for merging independent belief pieces of evidence collected from different agents. DST has been applied in many fields, including decision making, information fusion, pattern recognition, and sensor fusion [112]. In his mathematical theory of evidence, Shafer introduced the theoretical framework of belief functions and the so-called Dempster-Shafer's rule of combination to merge sources of evidence. In the following, we present the main notions of DST.

Let $\Theta = \{\theta_i, i = 1, ..., n\}$ be the *frame of discernment* of a problem under consideration. The set Θ consist of n exhaustive and exclusive possible values for a state variable v of interest. The powerset of Θ, denoted as 2^Θ, represents the set of all subsets of Θ. A *basic belief assignment* (BBA) m on Θ [109, 36, 62]., also called a belief mass function, is a mapping of the powerset 2^Θ to $[0, 1]$ that satisfies the following conditions:

$$m(\emptyset) = 0 \; and \sum_{X \subseteq \Theta} m(X) = 1, \tag{2.2}$$

where the values $m(X)$of a BBA are called *basic belief masses* and represent how strongly the evidence supports X. If $m(\emptyset) = 0$, m is said to be normal and if and only if $m(X) > 0$, each subset $X \subseteq 0$ is called a focal element of m. A BBA m can also be represented by its associate belief function *Bel* and plausibility function *Pl* respectively, defined as follows:

$$Bel(X) = \sum_{Y \subseteq X \neq \emptyset} m(Y) \; and \; Pl(X) = \sum_{Y \cap X \neq \emptyset} m(Y). \tag{2.3}$$

In contrast to probability theoretic approaches, the DST facilitates the expression of the belief in every element of 2^Θ and not only the elementary hypotheses. Thus, even information sources can be used that can only provide their knowledge about subsets of Θ. Thus, it is feasible to fuse two BBAs received from sources S_1 and S_2 with different reliability using Dempster's rule of combination:

$$m_{S_1 \oplus S_2}(X) = m_{S_1}(X) \oplus m_{S_2}(X) = \frac{1}{1-k} \sum_{A \cap B = X} m_{S_1}(A) m_{S_2}(B), \tag{2.4}$$

where k is a normalization constant representing the *degree of conflict* between m_{S_1} and m_{S_2} defined as:

$$k = \sum_{A \cap B = \emptyset} m_{S_1}(A) m_{S_2}(B). \tag{2.5}$$

The use of Dempster's rule is mathematically possible only if m_{S_1} and m_{S_2} are not totally conflicting, and the normalization constant k reflects the degree of conflict between the two sources. This normalization effectively redistributes the conflicting belief masses to the non-conflicting ones, hereby eliminating the conflict between the sources. Dempster-Shafer's

rule of combination is associative, commutative and nonidempotent [62]. However, the rule cannot be applied if the two sources are in complete opposition. These properties of Dempster-Shafer's rule of combination are beneficial for real-time applications as sources can be combined sequentially, and at a random order [36]. Moreover, the functions are simple to implement and compute. However, these results should be considered with caution, as a later study [36] showed that the order of at which sources get aggregated may impact the results. Another downside of the DST is that it can lead to counter-intuitive results when combining conflicting sources [128]. The limitations of the Dempster-Shafer theory are addressed in the Subjective Logic theory proposed by Jøsang [61, 60].

2.3 Subjective Logic Theory

When we assume an objective world, we can use binary logic to assert propositions about a state of the world to be either false or true. However, the world is unpredictable, and in many situations, one cannot determine the nature of a proposition with certainty. In other words, it is practically impossible to determine with absolute certainty whether a given proposition is true or false. Through probability calculus, which takes argument probabilities in the range [0,1], we are able to reflect subjectivity by allowing propositions to be partially true. However, due to the lack of sufficient evidence, we are often unable to estimate probabilities with confidence. Furthermore, whenever the truth of a proposition is assessed, it is always done by an individual, and it cannot be considered to represent a general and objective belief. In order to reflect as faithfully as possible the perceived world in which we are immersed, a formalism to express degrees of uncertainty about beliefs is needed; said formalism also shall include belief ownership to reflect the subjective nature of beliefs [61, 60].

Subjective Logic (SL) [61, 60] is a framework for artificial reasoning based on probabilistic logic and Dempster-Shafer theory of evidence [34, 109]. The idea of explicit representation of ignorance and fusing sources of evidence are inherited from the Dempster-Shafer belief theory [60, 109], and the interpretation of an opinion in Bayesian perspective is possible by mapping opinions into probability distributions [60]. To reason with propositions whose truth-values are uncertain, Bayesian probability and statistics can also be employed [45]. As previously explained in Section 2.2.1, Bayesian Probability is an interpretation of the concept of probability that can be seen as an extension of propositional logic. In Bayesian statistics, probability values are used as the fundamental measure of uncertainty. Therefore, it allows for reasoning with propositions whose truth-values are uncertain [45]. However, this type of probabilistic logic does not allow to seamlessly model situations where different agents express their beliefs about the same proposition. SL explicitly integrates the subjective nature and ownership of beliefs in its formalism, allowing the combination of different beliefs about the same proposition. Additionally, as mentioned above, SL is based on the Dempster-Shafer Theory of evidence (DST). In particular, Dempster-Shafer's rule of combination, originally proposed for merging sources of evidence in DST, is also used in SL.

To summarize, in recent years, SL has gained prominence because of its capability to deal

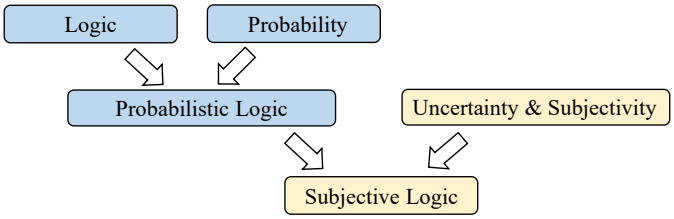


Figure 2.2: *Subjective Logic Framework from [61].*

(a) *Example of binary domain.* **(b)** *Example of n-ary domain.*

Figure 2.3: *Domains example from [61].*

with the degree of (un)certainty of propositions, i. e., it explicitly represents the amount of "uncertainty on the degree of truth about a proposition" [60]. Concretely, SL inherently allows 1) uncertainties representation as part of the fundamental building block of SL, called *Subjective Opinions* (further explained in Section 2.3.1), and 2) reasoning about the uncertainties through a process of *Belief Fusion* in which multiple Subjective Opinions are aggregated based on the selected fusion operator (further explained in Section 2.3.3).

2.3.1 Subjective Opinions

The fundamental building block of SL is a *subjective opinion* that represents the amount of uncertainty on the degree of truth about a proposition. The representation of a subjective opinion is a composite function consisting of belief masses, uncertainty mass and base rate. An opinion expresses a belief about the state of a variable which takes its values from a domain (i. e., a state space).

Domains

In SL, a state space consisting of a set of values is called a *domain*. A domain represents the possible states of a variable situation. The different values of a domain are exclusive i. e., only one state value is possible at any moment in time-, and exhaustive i. e., all possible state values are included in the domain. Domains can be binary (exactly two values) or n-ary (n values) where $n > 2$. A binary domain can be denoted $\mathbb{X} = \{x, \overline{x}\}$, where $\overline{x}$ is the complement (negation) of x, as illustrated in Fig. 2.3a.

Binary domains are typically used when modelling situations that have only two alternatives, such as a location in a map in which can be task on or not. Situations with more than two alternatives have n-ary domains where $n > 2$. As an example, the domain $\mathbb{Y} = y_1, y_2, y_3, y_4$ represents a quaternary domain, illustrated in Fig. 2.3b.

The values of an n-ary domain are considered to represent a single possible state or event, such values are called singletons. It is possible to combine singletons into composite values. Let us assume a ternary domain $\mathbb{X} = x_1, x_2, x_3$. The hyperdomain of $\mathbb{X}$ is the reduced powerset denoted $\mathscr{R}(\mathbb{X})$ and illustrated in Fig. 2.4. The solid circles denoted x_1, x_2 and x_3 represent singleton values, and the dotted oval shapes denoted $(x_1 \cup x_2)$, $(x_1 \cup x_3)$ and $(x_2 \cup x_3)$ represent composite values.

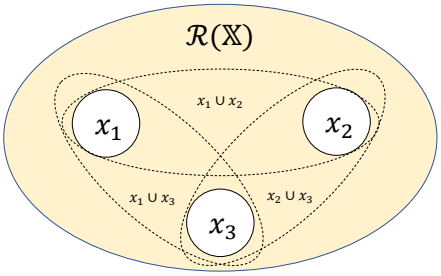


Figure 2.4: *Example hyperdomain from [61].*

Definition 1 (Hyperdomain [61])**.** Let $\mathbb{X}$ be a domain, and let $\mathscr{P}(\mathbb{X})$ denote the powerset of $\mathbb{X}$. The powerset contains all subsets of $\mathbb{X}$, including the empty set $\{\emptyset\}$, and the domain $\{\mathbb{X}\}$ itself. The hyperdomain denoted $\mathscr{R}(\mathbb{X})$ is the reduced powerset of $\mathbb{X}$, i. e., the powerset excluding the empty-set value $\{\emptyset\}$ and the domain value $\{\mathbb{X}\}$. The hyperdomain is expressed as

$$\text{Hyperdomain: } \mathscr{R}(\mathbb{X}) = \mathscr{P}(\mathbb{X}) \backslash \{\mathbb{X}\}, \{\emptyset\}.$$

Random Variables

Let $\mathbb{X}$ denote a binary or an n-ary domain. Then we can define X to be a random variable which takes its values from $\mathbb{X}$. For example, if $\mathbb{X}$ is a ternary domain, then '$X = x_3$' means that the random variable X has value x_3, which is typically interpreted in the sense that x_3 is TRUE. As a convention, domains are denoted by blackboard letters such as $\mathbb{X}, \mathbb{Y}$ or $\mathbb{Z}$, and that variables are denoted by italic capital letters such as X, Y or Z.

Let $\mathbb{X}$ be a ternary domain, and consider $\mathbb{X}$'s hyperdomain denoted $\mathscr{R}(\mathbb{X})$. From de definition of a hyperdomain we infer the possibility of assigning values of the hyperdomain to a variable. For example, it must be possible for a variable to take the composite value $x_1, x_3 \in \mathscr{R}(\mathbb{X})$, i. e., the real TRUE value is either x_1 or x_3, but it is unspecified which value in particular it is. Variables that take their values from a hyperdomain are called hypervariables.

Definition 2 (Hypervariable [61])**.** Let $\mathbb{X}$ be a domain with corresponding hyperdomain $\mathscr{R}(\mathbb{X})$. A variable X that takes its values from $\mathscr{R}(\mathbb{X})$ is a hypervariable.

A hypervariable X can be constrained to a random variable by restricting it to only take values from the domain $\mathbb{X}$. For simplicity of notation, we use the same notation for a random variable and for the corresponding hypervariable so that, e.g. X can denote both a random variable and a hypervariable. When either meaning can be assumed, we simply use the term variable.

Belief Mass Distribution and Uncertainty Mass

Subjective opinions are based on belief mass distributions over a domain $\mathbb{X}$, or over a hyperdomain $\mathscr{R}(\mathbb{X})$. In the case of multinomial opinions, the belief mass distribution is restricted to the domain $\mathbb{X}$. In the case of hyperopinions, the belief mass distribution applies to the hyperdomain $\mathscr{R}(\mathbb{X})$. Belief mass assigned to a singleton value $x_i \in \mathbb{X}$ expresses support for x_i being TRUE. The sum of belief masses can be less than one, i. e., belief mass distributions are sub-additive and their are complemented by uncertainty mass denoted u_X. In general, the belief mass distribution b_X assigns belief masses to possible values of the variable $X \in \mathscr{R}(\mathbb{X})$ as a function of the evidence support for those values. The uncertainty mass u_X represents the lack of support for the variable X to have any specific value. The sub-additivity of the belief mass distribution and the complement property of the uncertainty mass are expressed by the definition below.

Definition 3 (Belief Mass Distribution [61])**.** Let $\mathbb{X}$ be a domain with corresponding hyperdomain $\mathscr{R}(\mathbb{X})$, and let X be a variable over those domains. A belief mass distribution denoted b_X assigns belief mass to possible values of the variable X. In the case of a random variable $X \in \mathbb{X}$, the belief mass distribution applies to domain $\mathbb{X}$, and in the case of a hypervariable $X \in \mathscr{R}(\mathbb{X})$ the belief mass distribution applies to hyperdomain $\mathscr{R}(\mathbb{X})$. This is formally defined as follows.

$$\begin{aligned} &\textit{Multinomial belief mass distribution}: \ b_X : X \to [0,1], \\ &\textit{with the additivity requirement}: \ u_X + \sum_{x \in \mathbb{X}} b_X(x) = 1. \end{aligned}$$

$$\begin{aligned} &\textit{Hypernomial belief mass distribution}: \ b_X : \mathscr{R}(\mathbb{X}) \to [0,1], \\ &\textit{with the additivity requirement}: \ u_X + \sum_{x \in \mathscr{R}(\mathbb{X})} b_X(x) = 1. \end{aligned}$$

Base Rate Distributions

The default base rate of a composite value is equal to the number of singletons in the composite value relative to the cardinality of the whole domain. For example, given a domain $\mathbb{X}$ of cardinality k, the default base rate of each singleton value in the domain is $1/k$, and the default base rate of a subset consisting of n singletons is n/k. Default base rate is sometimes called *'relative atomicity'* in the literature. Base rates can also be dynamically updated. For example, when an urn contains balls of red (x) and black ($\overline{x}$) balls of unknown proportion, the initial base rates of the two types of balls can be set to the default base rate $a(x) = a(x) = 1/2$. Then, after having picked a number of balls, the base rates can be set to the relative proportions of observed balls.

Base rates are expressed in the form of a base rate distribution denoted a_X , so that $a_X(x)$ represents the base rate of the value $x \in \mathbb{X}$. Base rate distribution is formally defined below.

Definition 4 (Base Rate Distribution [61])**.** Let $\mathbb{X}$ be a domain, and let X be a random variable in $\mathbb{X}$. The base rate distribution a_X assigns base rate probability to possible values of $X \in \mathbb{X}$, and is an additive probability distribution, formally expressed as:

$$\textit{Base rate distribution}: \ a_X : X \to [0, 1],$$

$$\textit{with the additivity requirement}: \ \sum_{x \in \mathbb{X}} a_X(x) = 1.$$

Integrating base rates with belief mass distributions enables a better and more intuitive interpretation of opinions, facilitates probability projections from opinions, and provides a basis for conditional reasoning. When using base rates for probability projections, the contribution from uncertainty mass is a function of the base rate distribution.

The base rate distribution is normally considered to be general (i. e., not subjective) because it is based on common background information. Although different analysts can have different opinions on the same variable, they normally share the same base rate distribution over the domain of a particular situation. Nonetheless, it is obvious that two different observers can assign different base rate distributions to the same random variable in case they do not share the same background information. Base rates can thus be partially objective and partially subjective. This flexibility allows two different analysts to assign different belief masses as well as different base rates to the same variable. This naturally reflects different views, analyses and interpretations of the same situation by different observers.

The base rates associated to events of a frequentist nature (i. e., that can be repeated several times), can be derived from statistical observations. For events that can only happen once, the analyst must often extract base rates from subjective intuition or from analyzing the nature of the situation at hand and any other relevant evidence. However, in many cases, this can lead to considerable vagueness about base rates, and when nothing else is known, it is possible to use the default base rate distribution for a random variable. More specifically, when there are k singletons in the domain, the default base rate of each singleton is $1/k$.

The usefulness of base rate distributions is to make possible the derivation of projected probability distributions from opinions. Projection from opinion space to probability space removes the uncertainty and base rate to produce a probability distribution over a domain. The projected probability distribution depends partially on belief mass and partially on uncertainty mass, where the contribution from uncertainty is weighted as a function of the base rate.

2.3.2 Binomial Opinions Representation

In general, the notation ω_X^C is used to denote opinions in subjective logic, where the subscript X indicates the target variable or proposition to which the opinion applies, and the superscript C indicates the subject agent/source who holds the opinion. Superscripts can be omitted when it is implicit or irrelevant who the belief owner is. A binary domain consists of only two values and the variable is typically fixed to one of the two values.

Formally, let a binary domain be specified as $\mathbb{X} = \{x, \overline{x}\}$, then a binomial random variable $X \in \mathbb{X}$ can be fixed to $X = x$. Opinions on a binomial variable are called binomial opinions, and a special notation is used for their mathematical representation.

Definition 5 (Binomial Opinion [61])**.** Let $\mathbb{X} = \{x, \overline{x}\}$ be a binary domain with binomial random variable $X \in \mathbb{X}$. A binomial opinion about the truth/presence of value x is the ordered quadruplet $\omega_x = (b_x, d_x, u_x, a_x)$, where the additivity requirement

$$b_x + d_x + u_x = 1$$

is satisfied, and where the respective parameters are defined as
b_x : belief mass in support of x being TRUE (i.e. $X = x$),
d_x : disbelief mass in support of x being FALSE (i.e. $X = \overline{x}$),
u_x: uncertainty mass representing the vacuity of evidence,
a_x : base rate, i.e., prior probability of x without any evidence.

The projected probability of a binomial opinion about value x is defined by:

$$P(x) = b_x + a_x u_x$$

A binomial opinion can be visualized as a point in a barycentric coordinate system of three axes represented by a 2D simplex, which is in fact, an equilateral triangle, as illustrated in Fig. 2.5. Here, the belief, disbelief and uncertainty axes go perpendicularly from each edge towards the respective opposite vertices denoted x, $\overline{x}$ and u, respectively. The base rate a_x is a point on the baseline (also called probability axis). The line connecting the top angle of the triangle and the base rate point is called the base rate director. The projected probability $P(x)$ is determined by projecting the opinion point onto the baseline, parallel to the base rate director. The binomial opinion $\omega_x = (0.40, 0.20, 0.40, 0.90)$ with probability projection $P(x) = 0.76$ is shown as an example.

2.3.3 Belief Fusion

Through a process of belief fusion, multiple opinions regarding the same proposition are merged or aggregated into a single, collective opinion. The resulting single opinion is assumed to represent the ground truth better than each opinion in isolation. The process of opinion fusion is illustrated in Fig. 2.6, where we can see how multiple distinct sources, e. g., denoted C_1, C_2, ... C_N, can produce different and possibly conflicting opinions $\omega_X^{C_1}$, $\omega_X^{C_2}$,...$\omega_X^{C_N}$ about the same variable X. Multi-source fusion consists of merging the different sources into a single source that can be denoted $\diamond(C_1, C_2, ...C_N)$, and mathematically fusing their opinions into a single opinion denoted $\omega_X^{\diamond(C_1,C_2,...C_N)}$ which represents the opinion of the merged sources. The merger function is denoted by the symbol $\diamond$ [64]. A source of evidence could be, for e. g., a trust relationship between two actors in a peer-to-peer system, information of an observed system's event or a sensor in a robot which can be represented as an opinion. Depending on the specific application, a suitable operator of belief fusion with different properties is required.

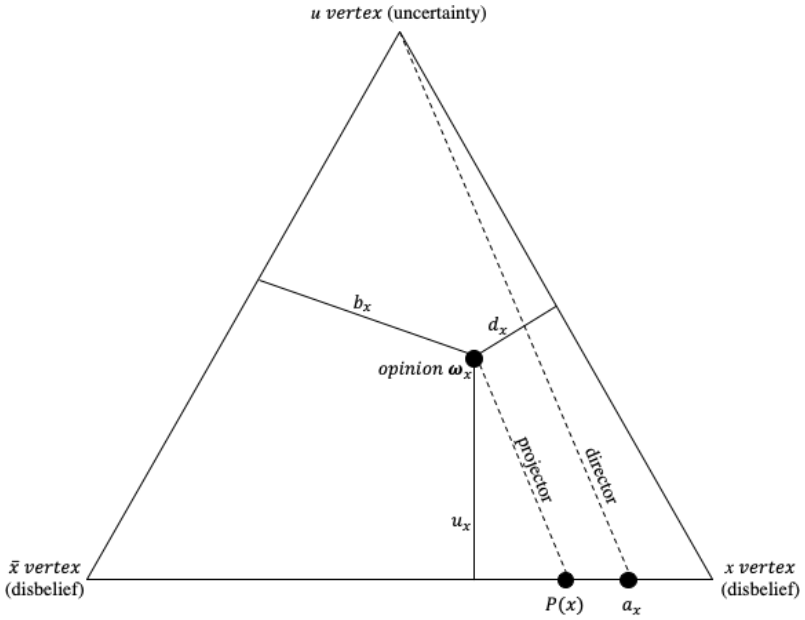


Figure 2.5: *Visualization of an binomial opinion from [61].*

Subjective Logic Operators

SL is a generalization of binary logic and probability calculus. As a result, most of the operators are generalizations and extensions of operators from binary logic and probability calculus, while others are unique to subjective logic. Concretely, the operators relevant for the Belief Fusion of different SL opinions are: *averaging belief fusion, cumulative belief fusion, weighted belief fusion, consensus & compromise fusion* and *belief constraint fusion.* Each of these fusion operations is designed to determine the shared belief and uncertainty of a group of evidence sources, with different applications depending on how evidence should be combined.

In the following, we give more details on each Belief Fusion operator:

- **Belief Constraint Fusion** (BCF) merges the preferences of multiple agents to find the preference of the group [61]. However, when opinions disagree, a common preference cannot be found, and belief fusion is not possible. As an example, consider a group of friends that want to watch a movie together. If they all want to watch the same movie, they can agree; otherwise, an agreement is not possible.
- **Cumulative Belief Fusion** (CBF) treats the individual opinions that are aggregated as independent pieces of evidence for the same proposition. This cumulatively increases the belief and/or disbelief value of the aggregated opinion while reducing its uncertainty. It is most suitable for combining multiple non-conflicting opinions. For example, repeated measurements in an experiment can be fused to form an opinion with an uncertainty smaller than that of an individual measurement.
- **Averaging Belief Fusion** (ABF) calculates the average across all aggregated opinions. It is commonly used when the opinions are dependent on each other, such that a larger number of opinions does not represent more evidence for or against a

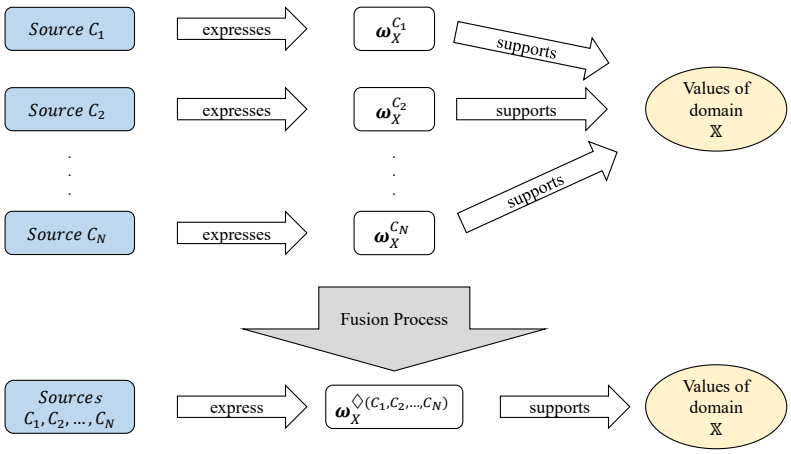


Figure 2.6: *Fusion process adapted [64].*

proposition. One example would be an examination board grading the dissertation of a student. Unlike other fusion operators, the outcome of this fusion process is affected by aggregating vacuous opinions.

- **Weighted Belief Fusion** (WBF) weights each opinion by the confidence in the opinion and then averages the result. As the uncertainty in the opinion rises the confidence decreases. The ABF is a special case of the WBF where all the uncertainties are equal. Due to this weighting, a vacuous opinion does not affect the result. An example of this type of situation is when, e. g., medical doctors express opinions about a set of possible diagnoses.
- **Consensus & Compromise Fusion** (CCF) maintains the shared belief masses between all the aggregated opinions. For conflicting opinions, a compromise is found, which has increased uncertainty. This operator is particularly helpful for identifying a set of shared beliefs among all agents. CCF is idempotent, commutative and considers a vacuous opinion as a neutral element. This operator is helpful in situations when different experts generate opinions identifying different options, such as when doctors with different expertise suggest potential causes in a diagnostic process. The fused opinion reflects the opinion of all experts and illustrates the group as a whole is certain about a certain set of potential causes.

Demonstration of the Belief Fusion with Different Operators

In Tabs. 2.2 and 2.3, we show a sample calculation to demonstrate the different belief fusion operators is shown for a pair of agreeing and disagreeing opinions ω_x^1, ω_x^2 on the same variable or proposition. respectively. The calculations were performed using the Subjective Logic Library[1].

Table 2.2 shows that when the opinions that are being aggregated agree, the CBF and BCF function create an aggregated opinion whose belief exceeds that of the sources. The

[1]https://github.com/vs-uulm/subjective-logic-java

difference between these two operators is that the CBF operator maintains increased the disbelief slightly with respect to ω_x^1, whereas the BCF operator actively reduces the disbelief below the ω_x^1 value. The WBF and CCF operators try to find compromises between opinions which is why their belief and disbelief values are in between the belief masses of ω_x^1 and ω_x^2. The table further demonstrates that the ABF operator is simply an average between the two source opinions. Similarly, one can see that the WBF is an average in which the opinions are weighted inversely by their uncertainty.

	Opinions		Aggregation Method				
Parameters	ω_x^1	ω_x^2	BCF	CBF	ABF	WBF	CCF
b_x	0.85	0.52	0.91	0.84	0.69	0.81	0.80
d_x	0.10	0.18	0.07	0.11	0.14	0.11	0.11
u_x	0.05	0.30	0.02	0.05	0.17	0.08	0.09
a_x	0.50	0.50	0.50	0.50	0.50	0.50	0.50
$P(x)$	0.88	0.28	0.92	0.86	0.77	0.85	0.84

Table 2.2: *Aggregating two concurring opinions ω_x^1, ω_x^2 using different belief fusion operators.*

	Opinions		Aggregation Method				
Parameters	ω_x^1	ω_x^2	BCF	CBF	ABF	WBF	CCF
b_x	0.85	0.20	0.73	0.72	0.53	0.71	0.35
d_x	0.10	0.64	0.25	0.24	0.37	0.22	0.14
u_x	0.05	0.16	0.02	0.04	0.10	0.07	0.51
a_x	0.50	0.50	0.50	0.50	0.50	0.50	0.50
$P(x)$	0.88	0.28	0.74	0.74	0.58	0.75	0.61

Table 2.3: *Aggregating two disagreeing opinions ω_x^1, ω_x^2 using different belief fusion operators.*

When the initial opinions are in conflict with each other, the BCF and CBF still try to maximize their belief masses. It is important to note that this process of aggregating conflicting opinions still reduces the uncertainty of the aggregated opinion when using the CBF and BCF operators, which is clearly undesirable. While the WBF operator does not actively reduce the uncertainty since it weighs opinions by their confidence, an initial opinion with a low uncertainty will result in an aggregated opinion with low uncertainty, even if the opinions are conflicting. In contrast, the CCF operator actively increases the uncertainty when states are conflicting. This is a stark difference between two operators that are used for finding compromises between conflicting opinions and should be considered when choosing the operator to resolve conflicts.

Selection of a Belief Fusion Operator

To select the adequate Belief Fusion operator, Jøsang [61] developed a set of guidelines summarized in Fig. 2.7 and further described below. Concretely Fig. 2.7 illustrates the decision process for selecting the most adequate belief fusion operator for the specific application.

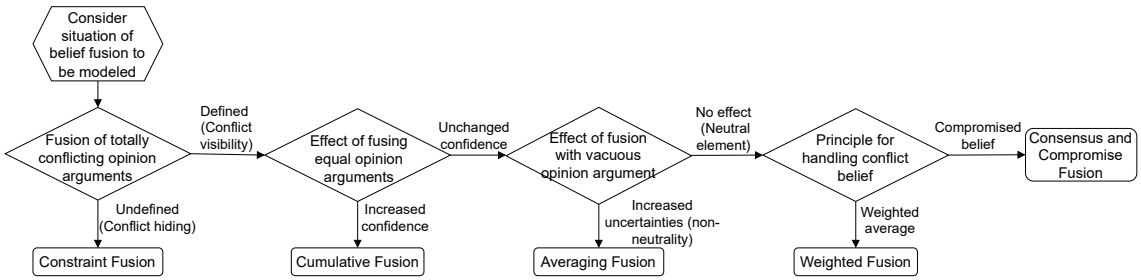


Figure 2.7: *Decision process for selecting the most adequate fusion operator, adopted from [61].*

1. If no compromise can be accepted between totally conflicting opinions, then it is probably adequate to apply the belief constraint fusion operator. This fusion operator reflects the assumption that there is no compromise solution for totally conflicting opinions.
2. In the case that equal arguments are considered as independent support for specific values of the variable so that equal opinions produce reduced uncertainty, then it is probably adequate to apply the cumulative fusion operator. This operator can also handle totally conflicting opinions.
3. In the case that a vacuous opinion has an influence on the fused result, then it is probably adequate to apply the averaging fusion operator. This can be meaningful where the lack of opinions (uncertainty) shall also be visible.
4. If the conflict between opinions must be solved, the simplest belief conflict management principle is to apply a weighted belief fusion operator. In case it is assumed that conflicting belief should be transformed into compromise belief, then it is probably adequate to apply consensus & compromise fusion. This operator takes into account common belief between belief arguments and is thereby preferable than weighted belief fusion.

In our MA-SACPS the observations are made independently by multiple agents (robots); hence, their opinions can be treated as independent pieces of evidence. Furthermore, compromises between said opinions are desired, such that the aggregated opinion is as accurate as possible. For the sake of accuracy, the compromise between conflicting opinions should only comprise of belief masses all agents can agree upon. Moreover, if the observations of different agents disagree, this should be reflected with a higher uncertainty. Lastly, vacuous opinions would indicate that an agent has made a meaningless measurement and should not impact the aggregated opinion. According to the guidelines discussed above, as well as the sample calculations, we selected Cumulative Belief Fusion (CBF) and Consensus & Compromise Fusion (CCF) operators to implement the reasoning and the knowledge aggregation in the adaptation logic as part of our self-adaptive CPS. Also our proposed knowledge aggregation and uncertainty mitigation method is applicable to systems that are composed of a single as well as multiple agents. The mathematical definition of all the operators is omitted as part of this section, but they can be found in [64, 113]. Finally, although we focus on these two fusion operators, the implementation of the system in this thesis support information aggregation with all the fusion operators.

Part II

Theoretical, Architectural, Methodological and Technical Solutions

3 Towards a Taxonomy of Autonomous Systems

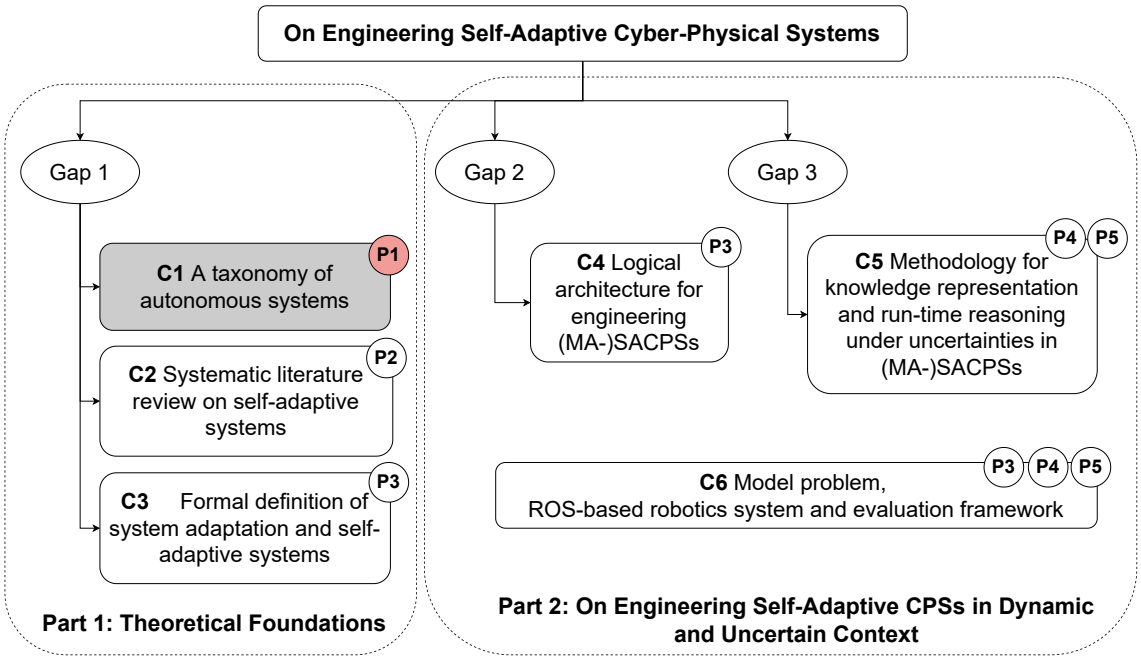


Figure 3.1: *Mapping of Gaps, Contributions and Publications to the structure of this thesis.*

Summary: The observed growing trend in putting the self-* properties of the systems in focus, concretely self-adaptation as a concrete point of interest in this dissertation, originates from the quest for increased system autonomy and the continuous engineering aim to decouple the user (precisely the user's control) from the systems. During our efforts to define self-adaptive systems, we realized that it is necessary to gain a more profound understanding of system autonomy before being able to scope and discuss self-adaptive systems. In response, in this paper, we proposed a taxonomy that supports the formal specification of different levels of autonomous systems. For each level of autonomy, we also propose a high-level architecture in which we exemplify the interaction between the end-user, the system, and the context. Our goal is to propose terminology that, if broadly accepted, can be used for more effective communication and comparison of autonomy levels in software-intensive systems that go beyond the well-known SAE J3016 for automated driving.

Problem: Similarly, as system adaptation and self-adaptive systems, the lack of shared understanding of the notion of autonomy makes it difficult for the works across various domains to be compared or even discussed since the same term is used with different

semantics. For instance, very often in the literature, Unmanned Aerial Vehicles (UAVs) are misleadingly referred to as autonomous, although very often, an end-user completely controls their flying operation.

Gap: To the best of our knowledge, there is no other work beyond the SAE J3016 standard for automated driving systems, which classifies different levels of system autonomy. However, this standard focuses on driving automation, and it is applicable only in the automotive domain. Additionally, there are very few other prior efforts in the literature to define system autonomy both informally [51] and formally [83, 10]. However, there is no other work that proposes a taxonomy and *formally* defines different levels of autonomy.

Method/Solution: In this paper, to define system autonomy, we differentiate between the system itself, the end-user of the system, and the context in which the system operates. We consider all three as separate entities in our architecture and formalisms. Concretely, to define different autonomy levels, we 1) put the system function in the main focus, concretely by treating autonomy as a property of a single function, and 2) investigate the degree of interaction that an end-user has with the system (i. e., the system function), and 3) emphasize the importance of considering the *learning* aspects in autonomous systems, especially when the systems operate in highly dynamic, uncertain, and unknown environments and when the user's control of the system reduces. As part of this work, we use the FOCUS [18] formal modeling notation.

Contribution: As part of this paper, we propose a taxonomy that supports formal specifications of different levels of autonomous systems, accompanied by a high-level architecture for each level, which exemplifies the interaction between the system, the end-user, and the context in which the system operates. In a nutshell, while neglecting the complexity of the system and its context and what this implies, we can say that the degree of autonomy is proportional to what portion of the logic from the end-user is shifted to the system as control logic.

Limitations: As part of this paper, we do not investigate the relationship between the levels of autonomy and self-* properties of the systems, e. g. self-adaptation. However, based on the findings and our contributions in this paper, we can conclude that (self-)adaptation is not possible without a certain degree of autonomy. Namely, a certain level of autonomy, i. e., the decoupling between the system (i. e., the managed element as part of a self-adaptive system) and the control logic (i. e., the adaptation logic as part of a self-adaptive system) is a precondition for self-adaptive systems. Simultaneously, (self-)adaptation is a precondition for higher autonomy since it enables the system to deal with various unanticipated changes and uncertainties. Nevertheless, a clear distinction between these two notions is still open. We close this gap with the contributions in the rest of the chapters of this dissertation, in which we define system adaptation and characterize, define and specify self-adaptive systems.

Author Contribution: All the authors together conceived the problem statement and conceptualised the idea, including the paper's main objective and general solution. S. Kugele and A. Petrovska developed the taxonomy levels and the formalisation. The paper was written by A. Petrovska and S. Kugele, and I. Gerostathopoulos reviewed and edited the later drafts of the paper.

Towards a Taxonomy of Autonomous Systems

Stefan Kugele✉[1][0000−0002−9833−4548], Ana Petrovska[2][0000−0001−6280−2461], and Ilias Gerostathopoulos[3][0000−0001−9333−7101]

[1] Technische Hochschule Ingolstadt, Research Institute AImotion Bavaria, Germany
Stefan.Kugele@thi.de
[2] Technical University of Munich, Department of Informatics, Germany
ana.petrovska@tum.de
[3] Vrije University in Amsterdam, Faculty of Science, The Netherlands
i.g.gerostathopoulos@vu.nl

Abstract. In this paper, we present a precise and yet concise characterisation of autonomous systems. To the best of our knowledge, there is no similar work, which through a mathematical definition of terms provides a foundation for describing the systems of the future: autonomous software-intensive systems and their architectures. Such systems include robotic taxi as an example of 2D mobility, or even drone/UAV taxi, as an example in the field of 3D urban air mobility. The presented terms lead to a four-level taxonomy. We describe informally and formally the taxonomy levels and exemplarily compare them to the degrees of automation as previously proposed by the SAE J3016 automotive standard.

Keywords: Autonomous systems · Taxonomy · Architecture

1 Introduction

The world is changing, and so are systems. Woods [9] describes in his much-noticed article "Software Architecture in a Changing World" the evolution from monolithic systems back in the 1980s to intelligent connected systems in the 2020s. We share Woods's vision for future systems. Today's connected cyber-physical systems (CPSs) are not too far away from this vision. The missing link between the current systems and the autonomous systems that we outline for the future is twofold: First, systems will be capable of adapting their structure and behaviour in reaction to changes and uncertainties emerging from their environment and the systems themselves [4,6,8] – they will be adaptive systems. Second, they will be able to derive knowledge themselves during their operational time to infer actions to perform.

The modern CPSs, such as cooperative robotic systems or intelligent transportation systems, are per se distributed. The not too distant future probably brings hitherto unrivalled levels of human-robot interaction. In such scenarios, machines and humans share the same environment, i. e., operational context [1, 4]. Examples for those shared environments are (i) production systems (cf. Industry 4.0) or (ii) intelligent transportation systems with both autonomous

and human-operated mobility. As a result, autonomous behaviour becomes an indispensable characteristic of such systems.

The lack of shared understanding of the notion of autonomy makes it difficult for the works across various domains to be compared or even discussed since the same term is used with different semantics. For example, very often in the literature, Unmanned Aerial Vehicles (UAVs) are misleadingly referred to as autonomous, although an end user completely controls their flying operation. As another example, we take robots operating in a room, which use Adaptive Monte Carlo Localisation (AMCL) to localise themselves and navigate in the space. Even though the robots localising and navigating independently in the room is some form of autonomy, they simply cannot be called *fully autonomous systems* if they operate in a room in which they often collide or get in deadlocks. In these situations, human administrators need to intervene in order for the robots to be able to continue with their operation. The intervention from a user (i. e., human administrator) directly affects the system's autonomy.

In response, we present in this paper our first steps towards a unified, comprehensive, and precise description of autonomous systems. Based on the level of user interaction and system's learning capabilities, we distinguish four autonomy levels ($\mathbf{A_0}$-$\mathbf{A_3}$): non-autonomous, intermittent autonomous, eventually autonomous, and fully autonomous. Our goal is to offer a precise and concise terminology that can be used to refer to the different types/levels of autonomous systems and to present a high-level architecture for each level.

The remainder of this paper is structured as follows. In Sect. 2 we briefly sketch existing efforts to formalise autonomy and explain the formal notation we are using later on. In Sect. 3, we present our taxonomy. Finally, in Sect. 4, we discuss and conclude the paper and outline our further research agenda.

2 Background

2.1 Existing Efforts to Formalise Autonomy

An initial effort in the literature to formally define autonomy was made by Luck and d'Inverno [5]. In this paper, the authors argue that the terms agency and autonomy are often used interchangeably without considering their relevance and significance, and in response, they propose a three-tiered principled theory using the Z specification language. In their three-tiered hierarchy, the authors distinguish between objects, agents, and autonomous agents. Concretely, in their definition of autonomy, as a focal point, the authors introduce *motivations*—"higher-level non-derivative components related to goals." Namely, according to their definition, autonomous agents have certain motivations and some potential of evaluating their own behaviour in terms of their environment and the respective motivations. The authors further add that the behaviour of the autonomous agent is strongly determined by and dependent on different internal and environmental factors. Although the authors acknowledge the importance of considering different internal and environmental (i. e., contextual) factors while defining autonomy, in their formalisms, the importance of the user in defining autonomy is

entirely omitted. On the contrary, in our paper, we put the strongest emphasis on the user. Concretely, how the involvement of the user in the operation of the system diminishes, proportionally to the increase of the system's autonomy. We define levels of system's autonomy by focusing on the system's function and how much from the user's logic is "shifted" to the system in the higher levels of autonomy. We further touch on the importance of *learning*, especially when 1) the systems operate in highly dynamic, uncertain and unknown environments, and 2) the user's control on the system reduces. To the best of our knowledge, there is no prior work that defines different levels of autonomy *formally*.

2.2 Formal Modelling Approach

Within this paper, we use the formal modelling notation FOCUS introduced by Broy and Stølen [2]. We restrict ourselves to only those concepts necessary for the understanding of this work. In FOCUS, systems are described by their (i) *syntactic* and their (ii) *semantic* interface. The semantic interface of a system is denoted by $(I \triangleright O)$ indicating the set of *input* and *output channels*, $I, O \subseteq C$, where C denotes the set of all channels. Systems are (hierarchically) (de-)composed by connecting them via channels. A *timed stream* s of messages $m \in M$, e. g. $s = \langle \langle m_1 \rangle \ \langle\rangle \ \langle m_3 \ m_4 \rangle \dots \rangle$, is assigned to each channel $c \in C$. The set of timed streams $\mathcal{T}(M)$ over messages M associates to each positive point in time $t \in \mathbb{N}^+$ a sequence of messages M^*, formally $\mathcal{T}(M) = \mathbb{N}^+ \to M^*$. In case of finite timed streams, $\mathcal{T}_{\mathsf{fin}}(M)$ is defined as: $\mathcal{T}_{\mathsf{fin}}(M) = \bigcup_{n \in \mathbb{N}} ([1\colon n] \to M^*)$. In the example given, in the first time slot, $\langle m_1 \rangle$ is transmitted; in the second time slot, nothing is transmitted (denoted by $\langle\rangle$), and in the third depicted time slot, two messages $\langle m_3 \ m_4 \rangle$ are transmitted. *Untimed streams* over messages M are captured in the set $\mathcal{U}(M)$ which is defined as $\mathcal{U}(M) = (\mathbb{N}^+ \to M) \cup \bigcup_{n \in \mathbb{N}} ([1\colon n] \to M)$, i. e., each time slot is associated with at most one message and there can be streams of finite length. By $\overrightarrow{C}$, we denote *channel histories* given by families of timed streams: $\overrightarrow{C} = (C \to \mathcal{T}(M))$. Thus, every timed history $x \in \overrightarrow{X}$ denotes an evaluation for the channels in C by streams. With $\#s$, we denote the number of arbitrary messages in stream s, with $m\#s$ that of messages m. For timed streams $s \in \mathcal{T}(M)$, we denote with $s{\downarrow}(t) \in \mathcal{T}_{\mathsf{fin}}(M)$ the finite timed stream until time t. The system's *behavioural function* (semantic interface) f is given by a mapping of *input* to *output histories*: $f\colon \overrightarrow{I} \to \wp(\overrightarrow{O})$.

3 A Taxonomy for Defining Autonomy

In this section, we first describe how autonomy of a system is related to autonomy of its functions, then present the main ideas behind our proposed taxonomy, and finally describe both informally and formally the different levels of autonomy.

3.1 Autonomy as a Property of Individual Functions

CPSs such as modern cars are engineered in a way to deliver thousands of customer or user functions. These are functions that are directly controlled by the user, or at least the user can perceive their effect. Switching on the radio, for example, results in music being played. This is a customer function. On the

other hand, there are functions, for example, for diagnosis or for offering encryption services, which the customer cannot control directly, of whose existence often nothing at all is known and whose effects are not visible to the user. Considering the above-mentioned, it is not trivial to classify a complete system as autonomous or non-autonomous. Instead, autonomy is a property of individual functions. Let us take a vehicle that drives autonomously. We assume that this system still offers the functionality to the passengers to choose the radio station or the playlist themselves. Thus, the CPS operates autonomously in terms of driving but is still heteronomous in terms of music playback. A similar argumentation applies, for example, to vehicles that are equipped with automation functions of varying degrees of automation, as considered in the SAE J3016 standard. For this system, as well as for other multi-functional systems, it is not meaningful to conclude from the autonomy of a single function, the autonomy or heteronomy of the whole system. Therefore, the commonly used term of an autonomous vehicle is too imprecise since the term autonomy refers exclusively to its driving capabilities. Hence, also the SAE proposes not to speak about "autonomous vehicles" but instead about "level [3, 4, or 5] Automated Driving System-equipped vehicles" (cf. [7], §7.2).

The only two statements that can be made with certainty are the following: (1) if all functions of a system are autonomous, then the system can also be called autonomous, and (2) if no function is autonomous, then certainly the system is not autonomous. Anything in between cannot be captured with precision. Single-functional systems are a special case. In such systems, the autonomy or heteronomy of the single function is propagated to the system. For the sake of illustrating our taxonomy on a simpler case, we will focus on single-functional systems in the rest of the paper.

3.2 Main Ideas Behind the Taxonomy for Autonomy

Our first main idea is to define autonomy levels of a system by focusing on the system's function and specifically by looking at the level of interaction that a user has with the system. Intuitively, the more user interaction is in place, the less autonomous the system is. "More user interaction" can mean both more *frequent* interaction and more *fine-grained* interaction. Actually, these two characteristics very often go hand in hand: consider, for instance, the case of a drone: it can be controlled with a joystick with frequent and fine-grained user interaction (lower autonomy); it can also be controlled via a high-level target-setting routine with less frequent and more coarse-grained user interaction (higher autonomy).

The second main idea behind our taxonomy is to distinguish between systems that learn and ones that do not learn. By learning, we mean that systems can observe both their context and user actions and identify behavioural patterns (e. g. rules or policies) in the observed data (e.g. by training and using a classifier). Such patterns can be used at run-time to reduce the amount of user interaction with the system gradually. Hence, the more capable a system is of learning behavioural patterns, the more autonomous it can become.

Finally, the third main idea is to define a system as autonomous within an assumed operational context. The assumed context can be narrow (e. g. a

drone operating in a wind range of 0-4 Beaufort) or very broad (e. g. a drone operating under any weather conditions). The specification of the context can also be uncertain or incomplete, i. e., the designers of the system might not be able to anticipate and list all possible situations that may arise under a specific context assumption. In any case, the more broad context is assumed, the harder it becomes for a system to reach high autonomy.

3.3 Taxonomy Levels

The four levels of autonomous systems in our taxonomy are shown in Fig. 1. Figure 2 shows the interaction between the user u, the context c, and the system s, as well as the (very high level) architecture of the system at each level in the taxonomy.

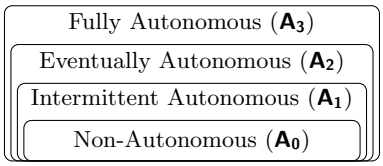
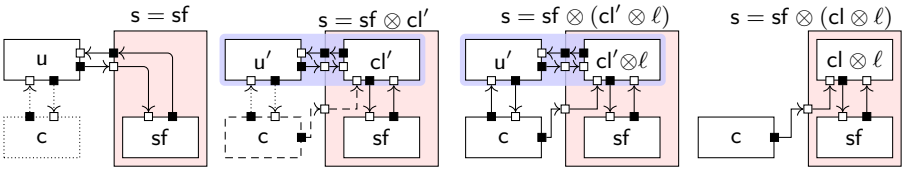


Fig. 1: Taxonomy levels.

The lowest level, $\mathbf{A_0}$, refers to systems that are *not autonomous*. For these systems, user input is needed at all times for controlling their operation. Examples are using the radio in a car or controlling the movement of a robot via a remote controller. As can be seen in Fig. 2(a), on this level, the system s (i. e., the system function sf) is completely controlled by the user and does not assume any input from the context (although this input might be already taken indirectly into account by the user). Note that the function sf might internally do something in the background that does not depend on the user input. A user can control the movement and trajectory of a drone; however, each drone internally provides attitude stabilisation that is not dependent on user input but is part of this system function.

The next level, $\mathbf{A_1}$, refers to systems that are *intermittent autonomous*: they can operate autonomously in-between two consecutive user inputs. In this case, the system can receive user input periodically or sporadically. As shown in Fig. 2(b), part of the logic of the user is shifted to the system as a control logic cl', which interacts with the system function sf. Input to the control logic can also be provided by the context. For instance, consider the movement of a robotic vacuum cleaner: the system perceives its environment through its sensors (obtains context input) and operates autonomously until it gets stuck (e. g. because of an obstacle or a rough surface); at this point, a user is required to intervene to restart the robot or point it to the right direction.

Level $\mathbf{A_2}$, shown in Fig. 2(c), refers to *eventually autonomous* systems: here, the user interaction reduces over time until the system reaches a point where it does not require any user interaction (user control). For this to happen, the system's control logic cl' is usually enhanced and equipped with a learning component ℓ that is able to identify the user interaction patterns associated with certain system and context states. An example is a robotic vacuum cleaner that is able to learn how to move under different floor types (e. g. faster or slower) and avoid crashes that would necessitate user interaction. Clearly, the degree and sophistication of monitoring and reasoning on context changes and user actions is much higher than in intermittent autonomous systems.



(a) $\mathbf{A_0}$ **(b) $\mathbf{A_1}$** **(c) $\mathbf{A_2}$** **(d) $\mathbf{A_3}$**

Fig. 2: From user-operation to autonomy: (a) A human user u controls the system s (i. e., the system's function sf). (b) The control logic is divided between the user u' and the system cl', i. e., $\mathsf{u} = \mathsf{u}' \otimes \mathsf{cl}'$. (c) The control logic of the system cl' could be enhanced with a learning component ℓ to better address e. g. changes in the context c. (d) The control logic cl with the usually necessary learning component ℓ is entirely performed by the system itself.

Finally, level $\mathbf{A_3}$ refers to *fully autonomous* systems, where no user input is needed (except the provision of initial strategic or goal-setting information), as it can be seen in Fig. 2(d). Systems on this level of autonomy can observe and adjust their behaviour to *any* context by potentially integrating learning in their control logic cl. Please note that the necessity and the sophistication of the learning is proportionate to 1) the complexity and the broadness of the context, and 2) the specifications of the context in the systems, as previously explained in Sect. 3.2. For instance, a robotic vacuum cleaner can move in a fully autonomous way when its context is more simplistic and could be fully anticipated (e. g. prescribed environment that contains only certain floor and obstacle types). To achieve this, the system needs to be equipped with sensing and run-time reasoning capabilities to adjust its movement behaviour and remain operational without human interaction. However, the difficulty for the same system to remain fully autonomous increases proportionally to the complexity of its context. For example, the context can be dynamic in ways that could not be anticipated, resulting in uncertain and incomplete context specifications. Since the user on this level is entirely out of the loop, this would require new, innovative, and more sophisticated learning methods in the *fully autonomous* systems.

We note that one can also imagine relatively simple systems without context impact that are configured *once or not at all* by a user and then work *without any user interaction or learning* (e. g. an alarm clock); while these systems also technically fall under $\mathbf{A_2}$ or $\mathbf{A_3}$, they are less complex and sophisticated.

3.4 Formalisation of Taxonomy Levels

The intuitively described taxonomy levels are specified mathematically in the following. We denote with u the input stream from the user to the system.

Definition 1 (Non-Autonomous, $\mathbf{A_0}$). *A system is called* non-autonomous, *iff it solely depends on user inputs:* $\forall t \in \mathbb{N}^+ \colon \mathsf{u}(t) \neq \langle\rangle$.

If there is less and less intervention or input by users, this becomes necessary repeatedly; we speak of intermittent autonomy.

Definition 2 (Intermittent Autonomous, $\mathbf{A_1}$). *A system is called* intermittent autonomous*, iff user interaction is necessary from time to time (periodic or sporadic), i. e.:* $\forall t \in \mathbb{N}^+ \; \exists t', t'' > t, t', t'' \in \mathbb{N}^+, t' \neq t'' \colon \mathsf{u}(t') \neq \langle\rangle \wedge \mathsf{u}(t'') = \langle\rangle$.

We emphasised that *learning* is essential in order to reach even higher levels of autonomy. By learning, the system converges to a point t after which no user interaction is needed anymore. Such systems are called *eventually autonomous*.

Definition 3 (Eventually Autonomous, $\mathbf{A_2}$). *A system is called* eventually autonomous*, iff after time* $t \in \mathbb{N}^+$ *no user input or intervention is needed anymore to fulfil the mission goals:* $\exists t \in \mathbb{N}^+ \colon \forall t' > t \colon \mathsf{u}(t') = \langle\rangle$.

In other words, only a finite number n of messages were transmitted up to t and no further messages will be transmitted beyond that time: $\#\mathsf{u}{\downarrow}(t) = n$, with $n \in \mathbb{N}$. The smaller t is, the earlier the point of autonomy is reached. If this is already the case from the beginning, we speak of *fully autonomous* systems.

Definition 4 (Fully Autonomous, $\mathbf{A_3}$). *A system is called* fully autonomous *if no user interaction or intervention is necessary at all, i. e.,* $\forall t \in \mathbb{N}^+ \colon \mathsf{u}(t) = \langle\rangle$.

Eventual and full autonomy make strict demands on the ability to precisely perceive and analyse the context, and draw conclusions and learn from it. However, in many respects, it will probably not be possible to achieve them in the foreseeable future for a not highly restricted operational context. Reasons for this are manifold and include the limited ability to fully perceive and understand the context and be prepared for all conceivable operational situations. Therefore, let us now consider intermittent autonomy. Assume the case that every other time step (e. g. every second minute), there is user interaction on an infinite timed stream, see u_1 below. This results in an infinite number of interactions. In another case, there could be one interaction every millionth minute, as shown in u_2. These two cases are equivalent or indistinguishable by definition.

$$\mathsf{u}_1 = \langle\langle m\rangle\langle\rangle\langle m\rangle\langle\rangle \ldots \langle m\rangle\langle\rangle \ldots\rangle, \; \mathsf{u}_2 = \langle\langle m\rangle\langle\rangle^{10^6-1}\langle m\rangle\langle\rangle^{10^6-1} \ldots \langle m\rangle\langle\rangle^{10^6-1} \ldots\rangle$$

This is due to Cantor's concept of infinity. Intuitively, however, a system that depends on user input every two minutes acts less autonomously than a system that can operate for almost two years (1.9 years in u_2) independently. Therefore, intermittent autonomy extends from "almost" no autonomy towards "almost" eventually autonomy. The classification in this spectrum can be made more precise if we take a closer look at the frequency of user input. Because of the above discussion on infinity, we only consider prefixes of finite length of (in)finite streams, i. e., $\mathsf{u}{\downarrow}(t)$. Let $\alpha \in (0, 1)$ be the ratio between times without user input and the interval $[1; t]$, i. e., $\alpha = \langle\rangle\#\mathsf{u}/t$. The closer α gets to one, the more autonomous the system is.

4 Discussion and Conclusion

Comparison to SAE Levels (L0-L5) [7]. No driving automation (L0) refers to $\mathbf{A_0}$–*no autonomy*, L1/2 (driver assistance, partial driving automation) can be defined with the notion of intermittent autonomy–$\mathbf{A_1}$, conditional driving automation (L3), applies for $\alpha \approx 1$ in a limited operational context such as highway

autopilots. Finally, high driving automation (L4) and full driving automation (L5) are captured by our level $\mathbf{A_3}$, *full autonomy*. For both, different assumptions, w.r.t. the context or the operational design domain, need to be made.

Future Extensions. It would be relevant to investigate the relation between the higher levels of autonomy and *self-** properties (cf. [3]) of the systems, e. g. self-adaptation. In our current understanding, adaptivity is a precondition for a higher autonomy since it enables the system to deal with various unanticipated changes and uncertainties; however, a clear distinction and definition of these two notions is still open. Another open issue refers to the notion of messages exchanged in intermittent autonomous systems. We have tried to distinguish between two intermittent autonomous systems based on their frequency of message exchange, but the expressiveness of messages is also important. Not every message has to have the same "information content". It is a matter for future research and discussion whether this point can be captured using, e. g. Shannon's definition of information content (a limitation of this approach is the assumption of statistical independence and idempotence of messages). To what extent or when is this a permissible limitation is an open question.

Conclusion. In this paper, we proposed a taxonomy that supports the formal specification of different levels of autonomous systems. We have also proposed a high-level architecture for each level to exemplify the user, context, and system interaction. Our goal is to propose a terminology that, if broadly accepted, can be used for more effective communication and comparison of autonomy levels in software-intensive systems that goes beyond the well-known SAE J3016 for automated driving.

References

1. Broy, M., Leuxner, C., Sitou, W., Spanfelner, B., Winter, S.: Formalizing the notion of adaptive system behavior. In: ACM Symposium on Applied Computing (SAC). pp. 1029–1033. ACM (2009)
2. Broy, M., Stølen, K.: Specification and Development of Interactive Systems–Focus on Streams, Interfaces, and Refinement. Monogr. in Comp. Science, Springer (2001)
3. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
4. de Lemos, R., et al.: Software engineering for self-adaptive systems: A second research roadmap. In: Software Engineering for Self-Adaptive Systems. LNCS, vol. 7475, pp. 1–32. Springer (2010)
5. Luck, M., d'Inverno, M.: A formal framework for agency and autonomy. In: First International Conference on Multiagent Systems. pp. 254–260. The MIT Press (1995)
6. Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. ACM transactions on autonomous and adaptive systems (TAAS) **4**(2), 1–42 (2009)
7. Society of Automotive Engineers: Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles, SAE j3016 (2018)
8. Weyns, D.: Software engineering of self-adaptive systems. In: Handbook of Software Engineering, pp. 399–443. Springer (2019)
9. Woods, E.: Software architecture in a changing world. IEEE Softw. **33**(6), 94–97 (2016)

4 Defining Self-Adaptive Systems: A Systematic Literature Review

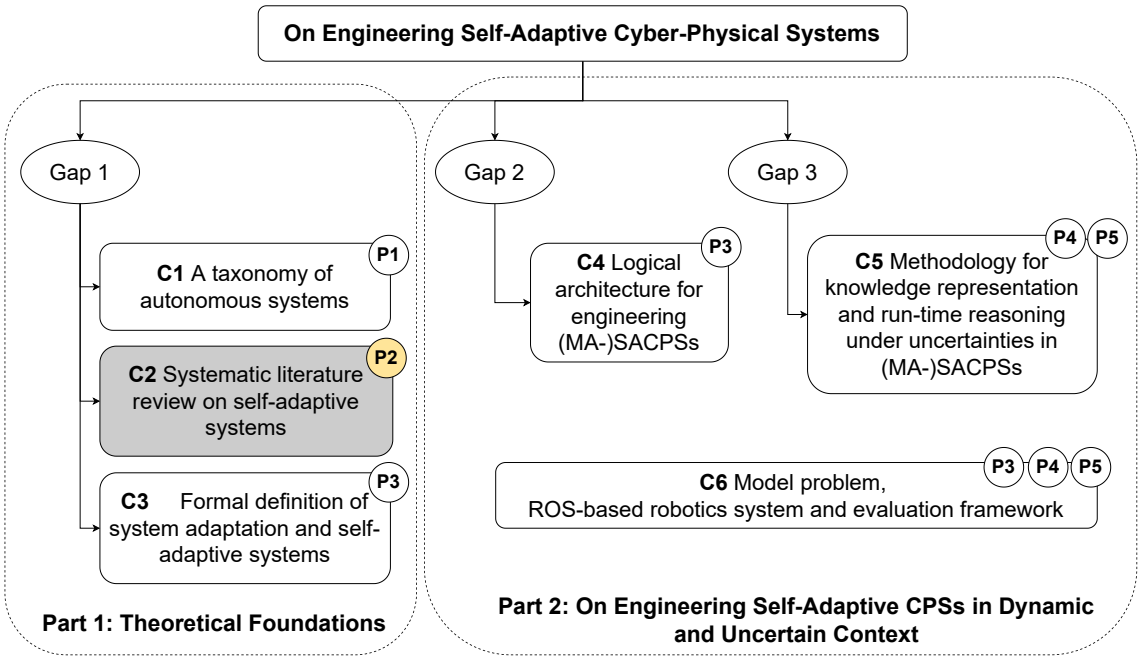


Figure 4.1: *Mapping of Gaps, Contributions and Publications to the structure of this thesis.*

Summary: In the last two decades since the IBM manifesto on autonomic computing was published, there has been an increased interest in self-* systems, including self-adaptive systems, which are the focus of this dissertation. However, despite the prominent research and the extensive work on self-adaptive systems, the literature still lacks precise, universally understandable and applicable definitions of these systems. We observed independently from other works in the literature [20, 82, 116] that so far, the terms system adaptation and self-adaptive systems have been primarily used intuitively without a precise explanation of their meaning. Although such usage might suffice in some instances, this is not the case in engineering and science, where a more rigorous definition is necessary [82]. In response, 1) to obtain an overview of the current state-of-the-art, 2) to gain insights into the limitations of the existing works, and 3) to set a foundation for a holistic formal definition, we conduct a systematic literature review focusing on how self-adaptive systems have been formally defined so far in the literature.

Problem: First and foremost, the lack of a concrete definition of self-adaptive systems hinders constructive scientific debates, which are impossible if experts have the same understanding of what self-adaptive systems are. Furthermore, the need for definitions

on *what* are self-adaptive systems has different engineering consequences, e. g. *how* to engineer these systems, that go beyond the famous MAPE-K conceptual model. Although MAPE-K gives some intuition behind the engineering of self-adaptive systems, primarily by the separation of concerns between the managed system and the managing system, a more specific semantics of these two components within the conceptual model is still lacking. Without more specific semantics accompanying the MAPE-K conceptual model, it is unclear how to separate between self-adaptive and self-* systems since MAPE-K is used as a conceptual model for engineering all the self-* systems. Similarly, the lack of precise semantics does not only affect the design phase of self-adaptive systems, but it also reflects in a lack of foundation on *how* to evaluate and compare these systems.

Gap: Over the years, various mapping studies and literature reviews have been published in the field of self-adaptive systems focusing on engineering self-adaptive systems [90, 84, 74, 119, 101]. However, to this date, no systematic study investigates how self-adaptive systems are formally defined in the literature that also summarizes their concrete limitations and aims to gain insight into why none of these formal efforts have been accepted by the community so far.

Method/Solution: As part of this contribution, we conduct a systematic literature review to answer "How are self-adaptive systems formally defined in the existing literature?", which is our leading research question as part of this paper. To support answering the leading research question in more detail, we derive three more refined research questions based on which we analyzed the primary studies and obtained our results:

1) Do the papers with formal definitions of self-adaptive systems also define *system adaptation* as part of their contributions?
2) Which characteristics of the self-adaptive systems are considered in the existing formal definitions and specifications?
3) Which formal notations have been used across different works to define self-adaptive systems?

Results: Our results reveal that despite the increasing interest in self-adaptive systems over the years, there is a scarcity of efforts that define these systems formally. Concretely, from an initial pool of 1493 papers, we have selected 314 relevant papers, which resulted in *only nine primary studies* whose objective was to define self-adaptive systems formally. Although it is not possible to define self-adaptive systems without first defining what it means for a system to adapt, our results show that out of these nine primary studies, only one work [23] aimed to define system adaptation as part of its contribution formally. The second unexpected insight from our results is that the notion of uncertainty is not considered in the formalisms in any of the primary studies, although uncertainty is considered the main reason for the need for self-adaptation across the literature. Finally, our results have shown that many of the primary studies provide their formalism by leveraging the aspects of collaboration [129], distribution [120], decentralisation [7] and ensembles [23] to define self-adaptive systems. However, system adaptation is not necessarily an emerging property

from collaboration or decentralization and should be defined in independence from these notions.

To summarize, a future formal definition of self-adaptive systems should provide more precise semantics by 1) defining what it means for a system to adapt and how system adaptation differs from system function, 2) considering more systematically all the different characteristics of self-adaptive systems in their formalism, in particular the aspect of uncertainty, and 3) defining adaptation and self-adaptive system in isolation from, e. g., collaboration and multi-agent systems. As a result, the prerequisite for specifying self-adaptive systems is first defining *system adaptation* and differentiating it from a nominal system functioning. Only after addressing and closing this research gap, we will be able to navigate toward a clear and precise definition of self-adaptive systems.

Contribution: Our study provides an overview of the current state-of-the-art research that has made efforts in the past toward proposing formal definitions of self-adaptive systems. Based on the analysis of the current research and its limitations, we elicit requirements and set a foundation for the future establishment of a holistic definition of self-adaptive systems. Having a more precise definition, or even in more general, a common, shared language and understanding of the core terminology of self-adaptive systems will complement the already existing works in this field, support scientific debates which currently lack a clear foundation for a discussion, and open new research directions for the future.

Limitations: Doing an automated search in the databases using the query "self-adaptive systems" yields hundreds of thousands of results. For that reason, we either searched in the databases by meta-data or only by title, depending on the advanced search options available in the concrete database. We assumed that if a paper defines self-adaptive systems, then that paper will certainly contain the keyword `(self-adapt*)` as part of these fields. However, this is not by any means a guarantee of completeness.

Author Contribution: A. Petrovska conceived the problem statement and conceptualised the idea and the mean objective of the paper. A. Petrovska also derived the exact methodology that was used for the systematic literature collection, selected the digital libraries for the collection of the papers, and created the exclusion and inclusion criteria used for filtering the initial set of papers. A. Petrovska and G. Erjiage jointly derived the searching query, and G. Erjiage executed the search and collected the initial set of papers. A. Petrovska and G. Erjiage did the filtering of the initial set of papers and the final selection of the primary studies. The writing of the paper was done by A. Petrovska, and S. Kugele provided review and editing on the final drafts of the paper.

Note: The paper is currently under review at the 18th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2023). For the coherent storyline of this thesis, the content of the paper is included in the following. Please note that the content of the paper is not included in the original template as submitted.

Defining Self-adaptive Systems: A Systematic Literature Review

Abstract

In the last two decades, the popularity of self-adaptive systems in the field of software and systems engineering has drastically increased. However, despite the extensive work on self-adaptive systems, the literature still lacks a common agreement on the definition of these systems. To this day, the notion of self-adaptive systems is mainly used intuitively without a precise understanding of the terminology. Using terminology only by intuition does not suffice, especially in engineering and science, where a more rigorous definition is necessary. In this paper, we investigate the existing formal definitions of self-adaptive systems and how these systems are characterised across the literature. Additionally, we analyse and summarise the limitations of the existing formal definitions in order to understand why none of the existing formal definitions is used more broadly by the community. To achieve this, we have conducted a systematic literature review in which we have analysed over 1400 papers related to self-adaptive systems. Concretely, from an initial pool of 1493 papers, we have selected 314 relevant papers, which resulted in nine primary studies whose primary objective was to define self-adaptive systems formally. Our systematic review reveals that although there has been an increasing interest in self-adaptive systems over the years, there is a scarcity of efforts to define these systems formally. Finally, as part of this paper, based on the analysed primary studies, we also elicit requirements and set a foundation for a potential (formal) definition in the future that is accepted by the community on a broader range.

4.1 Introduction

Since the publishing of the famous IBM manifesto on autonomic computing by Kephart and Chess [66] almost two decades ago, the interest in the self-* properties of the systems in software engineering has increased rapidly. Some of the most broadly spread and often found self-* properties in the literature are: self-adaptation, self-awareness, self-healing and self-organising, just to name a few. For example, the publications on self-adaptive systems have increased by 304% in the last twenty years, compared to the 50 years before that (from 1951-2001).[1]

There are many disciplines that have been considering the notion of *adaptation*, for example, biology and evolutionary sciences [16, 127], climate change and environmental sciences [89, 42], as well as film, cinematography and media studies [56]. The situation slightly differs in the field of software and systems engineering, where we can observe that the majority of the works available focus only on self-adaptive systems, without tacking and clarifying what is understood under the notion of adaptation in a first place. Hence,

[1] Source: ACM Digital Library.

defining the property of *system adaptation* is circumvented by the existing works, although defining what we understand under system adaptation is an essential prerequisite for a subsequent definition of self-adaptive systems.

Suppose we only focus on the available definitions of self-adaptive systems. In that case, we can observe the following: there exist prior works that propose informal definitions of self-adaptive systems as part of their papers [73, 32, 117]. However, all the informal definitions only rely on intuitive understanding communicated by the spoken language that is fairly ambiguous, which results in under-specified usage of the terminology of self-adaptive systems. In response, to tackle the limitations of the informal definitions, some researchers have put the focus on defining these systems formally [120, 22]. However, despite the notable advancements in the research on self-adaptive systems in the last two decades and the domain's active community, none of the existing formal definitions is broadly accepted and used as means of communication among the experts in the field. Therefore, the understanding of the core terminology still remains imprecise. To summarise, there is only an intuitive understanding of self-adaptive systems without a more profound understanding and a precise definition of these systems and how they differ from the "ordinary" systems considered non-adaptive. Furthermore, defining the property of *system adaptation* is the first step toward defining self-adaptive systems, and this is something that this research field has not paid enough attention to yet.

Other existing works in the literature also support our observations: Broy [20] and Lints [82] have independently reached the same concision regarding the intuitive use of the terms of adaptation and self-adaptive systems, arguing that although in some instances such intuitive usage might suffice, this is not the case in engineering and science, where a more rigorous definition is necessary [82]. Additionally, Weyns [116] in a recent work states that self-adaptive systems are not defined yet and that the lack of broadly accepted definitions is possibly the biggest challenge in the field of engineering self-adaptive systems [118, 119].

Problem. The lack of precise understanding of *what* are self-adaptive systems has different software engineering consequences and implications, for instance, *how* to build or engineer these systems that go beyond the famous MAPE-K conceptual model. The fundamental issue with the MAPE-K is that it serves as a reference model for engineering not only self-adaptive, but any self-* system in general. Although MAPE-K gives some intuition behind the engineering of self-adaptive systems, primarily by the separation of concerns between the managed system and the managing system, a more specific semantics of these two components within the conceptual model is still lacking. A more specific semantics accompanying the MAPE-K reference model, will also enable a better separation and characterisation of, e. g., self-adaptive, self-organising and self-aware systems.

Moreover, as mentioned before, besides the engineering implications, the lack of a concrete definition of self-adaptive systems has various scientific consequences. Namely, it hinders constructive scientific debates, which are impossible if experts have different understanding of what self-adaptive systems are. A better semantics of self-adaptive systems will 1) set a foundation for more constructive scientific debates, 2) complement the already existing works (methods, architectures, models, etc.) in this field, and 3) set the foundation on *how* to evaluate and compare these systems in the future.

Gap. Despite 1) the acceptance and the acknowledgement of adaptation as an emerging

property of software systems, and 2) the various systematic mapping studies and literature reviews in the field of self-adaptive systems [90, 84, 74, 119, 101], to the best of our knowledge, there is no other study that investigates and summarises how self-adaptive systems have been previously defined and characterised in the literature. In particular, no prior work summarises and analyses the existing formal definitions of self-adaptive systems in order to understand and gain insight into why none of these formal efforts is accepted by the community and what are their concrete limitations.

Solution. As a result, as part of this paper, we conduct a systematic literature review, which aims at summarising and analysing the existing works that formally define and specify self-adaptive systems. The following central research question leads our research:

> How are self-adaptive systems *formally* defined in the literature?

To tackle this broad research question, we derive three more refined research questions (further explained in Section 4.2), investigating 1) if the existing formal definitions also formalise the notion of *system adaptation* as part of their contributions, 2) which characteristics of self-adaptive systems are considered in the existing formalism, and 3) the formal notations used in each of the studies.

Contribution. Our systematic literature review provides an overview of the current state-of-the-art and structures the existing knowledge on how self-adaptive systems have been defined in the literature so far. More importantly, we analyse and summarise the limitations of the existing formal definitions, which provides new insights into why none of the formal definitions and specifications is accepted and used more broadly by the community. Our contributions also provide a foundation for improving the semantics of the core terminology of self-adaptive systems. This potentially leads towards a future establishment of a more unified understanding of these systems and, ideally, even to a broadly accepted definition of self-adaptive systems in the near future. A more profound understanding of the terminology will support the community in setting new challenges and identifying new directions for future research.

The rest of this paper is structured as follows: Section 4.2 presents the used methodology based on which we conduct our systematic literature review and the identified research questions. We present the results and answer the research questions in Section 4.3. In Section 4.4, we further discuss the main findings, followed up with a discussion on the limitations of this review. In Section 4.5, we present the related work and finally, Appendix A.6 concludes the paper.

4.2 Literature Review Methodology

This section describes the research methodology we followed in conducting the systematic literature review. The systematic process followed the guidelines proposed in various works by Kitchenham et al. [67, 68]. An overview of our complete methodology is presented in Fig. 4.2.

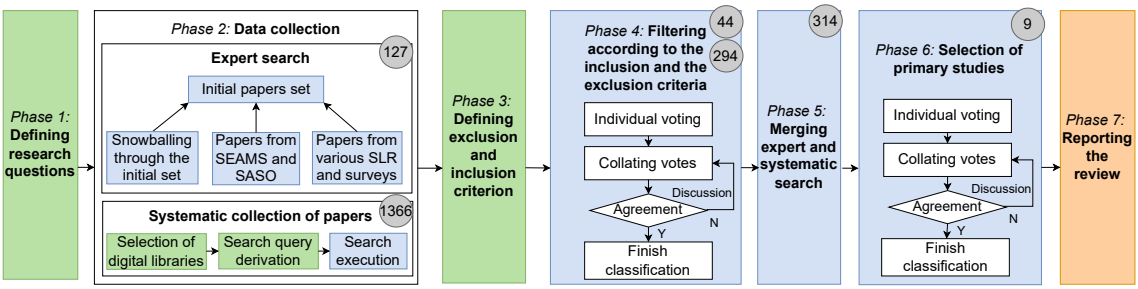


Figure 4.2: *Research methodology. With green, blue, and orange boxes, we depict the artefacts and the activities in the planning, conducting and reporting of the review.*

Phase 1: Defining research questions

The overall objective of the systematic literature review was to give an overview of the current state-of-the-art regarding the definition of self-adaptive systems in software and systems engineering, concretely **how self-adaptive systems have been *formally* defined in the existing literature**, which is *the leading research question* as part of this work. To support answering the leading research question in more detail, we derive three refined research questions:

RQ-A Do the papers with formal definitions of self-adaptive systems also define *system adaptation* as part of their contributions?

RQ-B Which characteristics of the self-adaptive systems are considered in the existing formal definitions and specifications?

RQ-C Which formal notations have been used across different works to define self-adaptive systems?

Phase 2: Data collection

In this study, we collected the papers in two different ways: manually by an expert search and by conducting a systematic studies collection.

Expert search. We initially started the data collection by collecting papers in a non-systematic way, referred to as an expert search. In the expert search, we started with an initial set of papers that included relevant studies based on our domain knowledge, known to us as key contributions in the field of self-adaptive systems. We extended this initial set of studies in three different ways. First, we snowballed through the related work of the initial set of studies, as described by Wohlin [125]. Second, we searched through the relevant papers in the conference proceedings of SEAMS[2] and SASO[3] as the two most relevant venues in this domain of research. Finally, we searched for relevant papers from previously published systematic literature reviews and surveys on self-adaptive systems. In this last step, we considered the studies from Weyns et al. [121], Muccini et al. [90],

[2]SEAMS stands for International Symposium on Software Engineering for Adaptive and Self-Managing Systems.

[3]SASO (currently ACSOS) stands for International Conferences on Self-Adaptive and Self-Organizing Systems.

Macías-Escrivá et al. [84], and Krupitzer et al. [73, 74]. At the end, our expert search resulted in 127 relevant studies in total.

Systematic studies collection. Our systematic search and collection of studies consist of two aspects: 1) *selection of digital libraries* on which we perform the automated search, and 2) *search query derivation*, which we later used as search queries in the selected databases.

We chose the following digital libraries to perform the search:

- ACM Digital Library (`https://dl.acm.org/`)
- IEEE Xplore (`http://ieeexplore.ieee.org/`)
- Scopus (`https://dl.acm.org/`)
- ScienceDirect (`http://www.sciencedirect.com/`)
- Wiley InterScience (`http://onlinelibrary.wiley.com/`)
- World Scientific (`https://www.worldscientific.com/`)

The search query derivation was an iterative process. Concretely, half a dozen of trial searches were performed in each database to evaluate the number of relevant studies obtained by different queries. Through this iterative process, we aimed to better understand the suitability of different search queries and keyword combinations, their advantages, and limitations, which was crucial for the final keyword query selection. Namely, we aimed for a search query as general as possible to consider a broad range of relevant papers from the literature while minimising the number of irrelevant studies. Some of the initial searching queries were the following: `(self-adapt* AND software)`, `(self-adapt* AND system)`, and `(self-adapt* AND engineer*)`. Our preliminary results showed that including the keywords `system` and `engineer*` in the query resulted in many irrelevant studies, e. g. from networks and hardware. On the opposite side, we realised that restricting only to the keyword `software` excludes works from the domain of cyber-physical systems, which are systems with increasing prominence in the field of self-adaptive systems in the last decade. Furthermore, since the main focus of this literature review is to get a better understanding of how self-adaptive systems are defined, we also tried using `(self-adapt* AND defin*)` as a searching query, which unfortunately gave only a few results. Different combinations of these searching keywords have led either to a broad set of irrelevant papers or to a very narrow search. For that reason, we used `(self-adapt* AND (software OR cyber-physical))` as a final searching query for our automated search on the databases identified above, searching by meta-data (title, abstract, keywords) or only by title, depending on the advanced search options available for the chosen databases. The systematic collection resulted in 1366 studies matching the derived searching query.

Phase 3: Defining inclusion and exclusion criteria

After the collection of the papers, we needed to perform the first study selection. Since we are exclusively interested in studies related to system adaptation and engineering self-adaptive systems, in this phase, we defined rigorous inclusion and exclusion criteria to filter the irrelevant papers collected during the extensive search in the previous phase.

Table 4.1: *Inclusion criteria.*

Criteria	Description
I1	Papers that have been published in conferences and journals, including full research papers, short papers, position papers, new ideas and emerging results papers, and papers from doctorate symposiums
I2	Literature published in book chapters
I3	Papers defining adaptivity, context, self-adaptivity in software engineering
I4	Papers proposing engineering approaches (for example, frameworks, methodologies, methods, reference architectures) for self-adaptive systems
I5	Papers focusing on modelling, design, architecture, and engineering of self-adaptive systems
I6	Systematic literature reviews and mapping studies on self-adaptive systems

The inclusion and the exclusion criteria that we defined for this purpose are presented in Tab. 4.1 and Tab. 4.2, respectively.

Phase 4: Filtering according to the inclusion and exclusion criteria

In this stage, we apply the inclusion and exclusion criteria to the studies collected through 1) the expert search, and 2) the automated systematic collection. Two of the co-authors of this literature review performed the filtering and selection of the studies in this stage. During the voting process, the title, the abstract, and, if necessary, the introduction and conclusion of each study (1493 in total: 127 from the expert search and 1366 from the systematic collection) were read and carefully examined to determine their relevance. The exact steps of the classification and the voting process of this phase are depicted in Fig. 4.2. In a nutshell, the authors voted and classified each paper individually. A discussion followed if the authors' votes were in disagreement until the authors reached a unified decision about the study under analysis. Applying the inclusion and the exclusion criteria resulted in 338 studies in total: 44 studies from the expert search and 294 studies from the systematic collection.

Phase 5: Merging expert and systematic search

In this phase, the filtered results from both the expert search and the systematic collection from the previous phase are combined, and the found duplicates are removed. This resulted in 314 unique and relevant papers.

Phase 6: Selection of primary studies

The selected relevant papers from the previous phase could be analysed in different ways based on the aims and the goals of the concrete study. Since in our work we are interested in how self-adaptive systems are *formally* defined in the literature, we analysed and classified the relevant studies from Phase 5 according to two questions, depicted in the activity diagram in Fig. 4.3. The selection process in this phase was similar to the inclusion

Table 4.2: *Exclusion criteria.*

Criteria	Description
E1	Papers that are not full research papers, including abstracts, tutorials, presentations, or lecture notes
E2	Papers without PDF and abstracts
E3	Tool papers, case studies, roadmaps, overviews
E4	Papers not focusing on self-adaptivity in software engineering and cyber-physical systems, but instead focus on
	E4.1 Specialised parts of software engineering, for example: software product lines, cloud-based, service-oriented
	E4.2 Other computer science fields: networking, hardware, middleware, OS, sensing, control and control theory or static robotics systems
	E4.3 Another field in general, for example energy, manufacturing, smart buildings, smart cities, traffic control, economics, natural processes etc.
E5	Papers not focusing on modelling, design, architecture, and engineering of self-adaptive systems, but instead on verification, validation, testing, or operation and maintenance phases of a software life cycle
E6	Papers not written in English

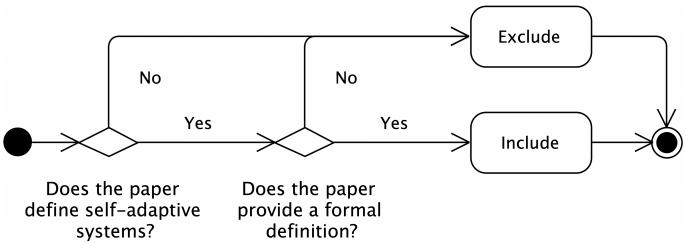


Figure 4.3: *Two-step selection process.*

and exclusion criteria filtering process described previously in Phase 4. In summary, two of the authors independently analysed and classified the 314 relevant studies from the previous step, according to the two-step selection process from Fig. 4.3. The votes were consolidated, and in case of disagreements, discussions took place among the authors until reaching a unified decision. Applying the two-step selection process in this phase resulted in a final set of *nine primary studies* that are analysed rigorously in the rest of this paper.

Please note the following about our analysis: 1) there were more than nine studies that included some formalism; however, we only selected those papers whose goal was to define self-adaptive systems formally and the studies with different objectives were excluded during this selection process, and 2) there were three more papers [54, 53, 52] that claimed to define self-adaptive systems in their abstract and introduction, but since the actual contributions of these papers did not fulfil their claims, we excluded them from our primary studies.

Phase 7: Reporting the review

A reproducible package with the selected studies in each of the phases of our methodology, the authors' voting and the analysed data is available online.[4] Additionally, the package contains the BibTeX bibliography (.bib) of all the relevant studies from Phase 5.

4.3 Results

4.3.1 General overview of the results

This section gives an overview of the 314 relevant papers analysed in Phase 6. In Fig. 4.4, we show the distribution of the papers over the years in different types of venues. We can also see in Fig. 4.4 that the first works on this topic were published in 1999, and the publication trend has grown since 2004, which can be correlated with two distinct events.

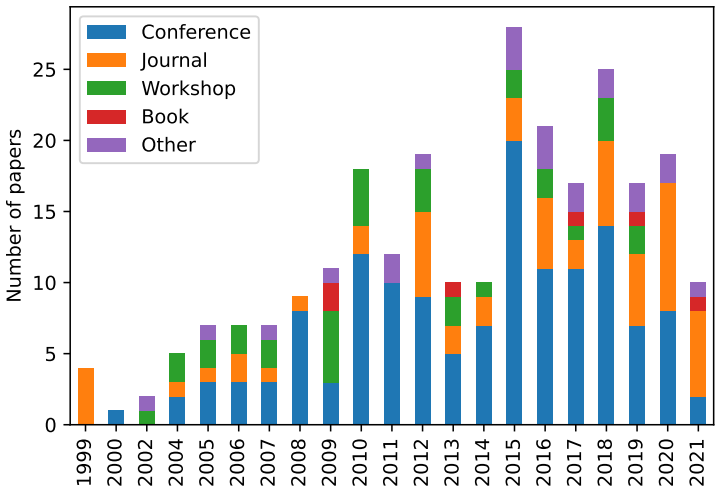


Figure 4.4: *Overview of the number of publications per year.*

The first event is related to the first noted instance of the term self-adaptive software in the literature in a technical report by Laddaga in 1997 [77]. In this report, Laddaga informally defines self-adaptive software systems as "[...] software that evaluates its own behaviour and changes behaviour when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible." The author also adds that the research in self-adaptive systems "[...] seeks a new basis for making software adaptive, that does not require specific adaptive techniques, such as neural networks or genetic programming, but instead relies on software informed about its mission and about its construction and behaviour." The second, probably even more significant event was the publishing of the famous IBM manifesto on autonomic computing by Kephart and Chess [66] in 2003. This paper introduced the MAPE-K conceptual model and set the foundation not only for engineering self-adaptive systems but self-* systems in general, e. g., self-awareness [81, 72] and self-healing [100, 49]. The manifesto on autonomic

[4] https://github.com/tum-i4/self-adaptive_SLR

computing also set a foundation for a whole new research field on self-adaptive systems, which has been expanding for the last two decades.

Figure 4.5 shows that from the 314 relevant studies we analysed in Phase 6, the majority of studies—56% of the studies (175 papers) provide neither an informal or formal definition nor an intuition of the authors' understanding of self-adaptive systems. We did not expect these results since these studies were rigorously selected to contribute with solutions for engineering self-adaptive systems. 41% of the studies (130 papers) provide informal definitions as part of their works, and *only 3% of the studies (9 papers) provide some formalisation* of the notion of self-adaptive systems. We selected those nine studies as primary for further analysis in our systematic review.

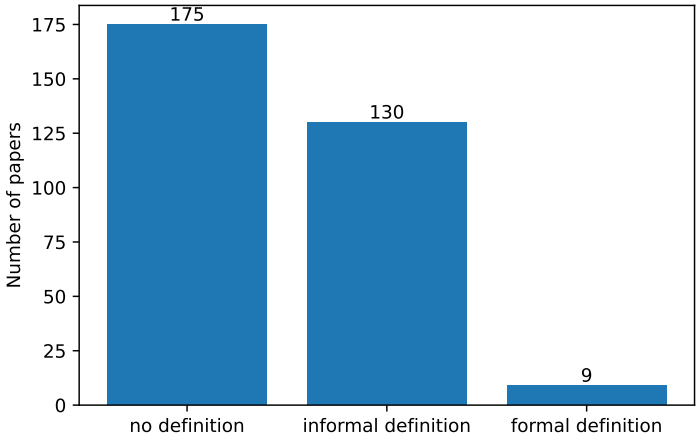


Figure 4.5: *Overview of the type of definitions.*

4.3.2 Identifying the different classes and dimensions for analysis

To answer the leading research question as part of this work and to discuss how self-adaptive systems are formally defined in the literature, we introduce four classes of analysis dimensions: **(C1)** papers that formally define the property of *system adaptation* as part of their formal definition of self-adaptive systems, **(C2)** papers that formalise MAPE behaviour, **(C3)** papers that consider different characteristics of self-adaptive systems in their formal definitions, and **(C4)** used formal notation. The introduced analysis classes contain eight analysis dimensions in total, based on which we analysed all the primary studies (see Tab. 4.3).

As discussed previously in Section 4.1, in order to define self-adaptive systems, we first need to understand what the notion of *system adaptation* means in the field of software and systems engineering. Defining adaptation as a system property is 1) the core pillar for defining self-adaptive systems, and 2) is necessary to compare the existing and future works in this field. Therefore, we want to investigate if the existing papers on formalising self-adaptive systems also define system adaptation as part of their contributions. Hence,

in our first class (C1), we differentiate between 1) papers with a concrete aim to *explicitly* formalise system adaptation, 2) papers that assume they define this notion *implicitly*, for instance, through formalising adaptive system behaviour, and 3) papers that *do not formally define* system adaptation in their work.

During our analysis, we also identified that some of the primary studies aimed at defining adaptive behaviour by specifying the behaviour of the **MAPE-K** feedback loop. In response, we introduced the second class (C2) for analysis.

In the third class (C3) of the analysis dimensions, we consider various characteristics identified in the literature as essential while defining self-adaptive systems based on the external and internal principles proposed in a recent work by Weyns [116]. As we elaborated previously, Weyns has stated that there is no consensus on the definition of self-adaptation so far in the community. In response to that, as part of his work [116], he proposes two complementary *principles*—external and internal—that characterise self-adaptive systems. The principles are built upon the consolidated usage of the notion of self-adaptive systems for the past decade in the community. To the best of our knowledge, this is the most complete consolidated characterisation of self-adaptive systems. For that reason, we used the characteristics from the principles to identify further dimensions for the analysis of our primary studies.

According to the external principle, a self-adaptive system handles changes and **uncertainties** from its environment (also referred to as **context**), the **system**, and the system goals autonomously. The context is the part of the environment relevant to a particular system [20]. These two terms have often been used interchangeably in the domain of self-adaptive systems; however, from our point of view, having a clear understanding and differentiation of context and environment is important. However, making this differentiation is not the aim of this work, and through the rest of the paper and in Tab. 4.3 we will use the concept of *context* only. By *system* in the table, we refer to the managed system that gains the ability to adapt as part of a self-adaptive system and not the self-adaptive system as a whole.

The internal principle separates the system goals in self-adaptive systems into domain and adaptation goals. The **domain goals** are related to the concerns of the managed system—the system that gains adaptation capabilities. Whereas the **adaptation goals** are related to the concerns of the managing system—the entity of the self-adaptive system that enabled the adaptation of the managed system. We use this differentiation from the principles and make a further semantic distinction between the managed and the managing system as part of a self-adaptive system. Concretely, we say that the domain goals are related to the functionality of the system—more precisely, to the fulfilment of the system function, i. e., the function of the managed system. In contrast to the domain goals, we consider the adaptation goals to be in relation to one or more quality criteria or objectives, which is also supported by prior works [114, 122]. In sum, we consider the separation of the goals in self-adaptive systems between domain and adaptation goals as essential, as it provides the basis for discussing and distinguishing when a system adapts and when it simply operates or functions. Furthermore, during the analysis of our primary studies, we did not concretely search whether these terms are used or not. Instead, we analysed the studies more thoroughly to answer if perhaps the studies adopt these ideas while using

different terminology, or if these ideas are implicitly considered in their contributions and formulas without giving them a specific name.

In a nutshell, in the third class (C3) of analysis dimensions, we differentiate between 1) papers that include a concrete characteristic *formally* as part of their definition, 2) papers that identify or mention a concrete characteristic only *informally* and do not include it as part of their formalism, and 3) papers that *do not* even identify or mention the necessity for the consideration of a concrete characteristic in their definition of self-adaptive systems.

Finally, in the fourth class (C4), we have noted the **formal notation** used in each paper.

4.3.3 Analysis of the primary studies

In this section, we analyse the primary studies in order to consolidate the existing work and answer the research questions. A thorough analysis of the primary studies and discussion of their limitations should enable us to set the foundation for improving the semantics and to derive requirements for a unified and precise definition of self-adaptive systems in the future. Although we collected the papers systematically, we ended up only with nine primary studies for the analysis. Therefore, we decided to take a more qualitative approach to analyse our primary studies guided by our leading research question that we previously introduced in Section 4.2). We summarise the qualitative analysis of our primary studies in the following, based on which Tab. 4.3 is filled. Due to space limitations, we are not giving the formal details, but we include the used formal notations in each of the primary studies.

One of the first efforts to formally define adaptive behaviour was made by Zhang and Cheng [129], in which the authors proposed a model-driven software development process for dynamically adaptive programs. According to the authors, adaptive programs are generally more difficult to specify due to their high complexity, especially in multi-threaded adaptations where the program behaviour results from the collaborative behaviour of multiple threads. This is the first main limitation of this work since adaptation is not necessarily an emerging property from a collaboration, and it should be treated and defined as a separate concept. In their formal representation of adaptive programs, a *program* is represented by a state machine that exhibits certain behaviour and operates in specific domains. A dynamically adaptive program operates in different domains and changes its behaviour (i. e., behavioural modes corresponding to the specific domain) at run-time in response to domain changes. As part of their work, the authors do not explicitly formalise system adaptation; however, they illustrate the specification process for three types of adaptive behaviour by modelling an audio streaming protocol with Petri nets. The authors use prior works on specifying dynamic systems architectures [5, 15] to formalise adaptive programs. As a result, they often use the terms adaptive and dynamic interchangeably throughout the paper without clearly distinguishing between them, which is the second limitation of this study. Lastly, this work does not consider any of the other analysis dimensions identified in class C3, which are paramount to be included in a holistic formal definition.

A similar concept in which adaptation is described through the realisation of different behavioural modes is proposed by Klarl [69]. In this work, the author realises the behavioural

modes by roles which can be dynamically adopted by a component. Concretely, the author proposes a model-driven engineering process to develop self-adaptive systems, in which the adaptation logic (i. e., the managing system) is considered independently from the application logic (i. e., the managed system) and supports the systematic transition between their components. For specification, the author proposes hierarchical adaptation automata, and for the design—a role-based architecture according to a Helena Adaptation Manager pattern. This study neither defines the notion of system adaptation nor adaptive behaviour. Except for considering the context (concretely, perceptions about the context) and the system state as attributes of the signature of self-adaptive component types in the formalism of the paper, no other analysis dimension from class C3 is considered as part of this study.

In two separate works, Broy et al. [20] and Bruni et al. [22] try to answer how the self-adaptive systems differ from the "ordinary" systems, which are considered non-adaptive. Concretely, Broy et al. aim at defining adaptive system behaviour while differentiating interaction patterns between three separate entities: the system, a subject (a user or other technical system that interacts with the system) and the context. The authors claim that one can differentiate the adaptive behaviour of the system only by considering and observing the context in which the system operates. The authors further classify the system inputs into direct/explicit and indirect/implicit, and assume that a system always receives the user inputs explicitly. Therefore, adaptive system behaviour can be observed if the system reaction resulting from the user input (the explicit input) is additionally determined by some additional information about the context received through the implicit inputs. Based on these ideas, the authors identify four types of observable system behaviour (i. e., adaptive behaviour) with respect to the user: non-adaptive, non-transparent adaptive, transparent adaptive and diverted adaptive behaviour. To summarise, as part of this work, the authors identify the consideration of the context and the system (state) as relevant and necessary for the system adaptation; and therefore, they include them as part of their formalism, which is based on FOCUS modelling approach.

Bruni et al. [22] propose a conceptual framework for adaptation, in which they assign a central role to control data, which governs the adaptive behaviour of a component. The authors define adaptation informally as a run-time modification of the control data and, consequently, consider a component as self-adaptive if it can modify its own control data at run-time. They formally define adaptable vs non-adaptable components, self-adaptive components, and knowledge-based adaptation, in which they recognise the context as the observable part of the environment. The authors formalise their conceptual framework using a Labelled Transition System (LTS) model. Similarly as in [20], the authors consider the context and the system state as part of their formalisation; however, all the other analysis dimensions from class C3 are not considered in either of these two works. The most significant shortcoming of this work is that the central idea of their concept (i. e., the control data) is left fuzzy and unclear since the authors do not elaborate precisely on what they understand under the notion of control data, how one can identify control data in the system, how the system is influenced by the control data and the structure of the control data. Furthermore, in contrast to the work by Broy et al. [20], Bruni et al. do not formalise or specify adaptive behaviour as part of their work.

Table 4.3: *Summary of papers that provide some formal definitions on system adaptation and self-adaptive systems.*

Class	Analysis Dimension	Zhang and B. H. C. Cheng [129], 2006	Broy et al. [20], 2009	Qureshi, Jureta, and Perini [103], 2011	Bruni et al. [22], 2012	Weyns, Malek, and Andersson [120], 2012	Arcaini, Riccobene, and Scandurra [7], 2015	Iglesia and Weyns [58], 2015	Klarl [69], 2015	Bucchiarone and Mongiello [23], 2019
C1	**System adaptation**	Implicit	Implicit	No	No	No	No	No	No	Explicit
C2	**MAPE behaviour**	No	No	No	No	No	Yes	Yes	No	No
C3	**Uncertainties**	No	No	No	No	No	No	No	No	No
	Context (state)	No	Formal	Formal	Formal	Formal	Formal	Formal	Informal	Formal
	System (state)	No	Formal	No	Formal	Formal	Formal	Formal	Formal	Formal
	Domain goals	No	No	Formal	No	No	No	No	No	No
	Adaptation goals	No	No	Formal	No	Informal	Informal	Formal	No	No
C4	**Formal notation**	Petri nets	FOCUS	Techne	LTS	Z language	ASM	TA, TCTL	LTS	TGG, LTS

Weyns et al. [120] propose formally specified models for designing self-adaptive software systems. The authors propose a FOrmal Reference Model for Self-adaptation (FORMS), which enables precise descriptions of the architectural characteristics of distributed self-adaptive software systems in the early design phases of the system. FORMS primarily focuses on the formalisation of the structural aspect of self-adaptive systems without providing any insights into the behavioural semantics of the self-adaptive systems. Although FORMS had and continues to have a notable impact in the community, it neither defines system adaptation nor adaptive behaviour. FORMS considers the aspect of context and system formally, and the adaptation goals are only considered informally throughout the work. Finally, similarly to [129], the authors of FORMS leverage some other concept—specifically in FORMS, the notion of system distribution—to compensate in some sense for the lack of precise understanding of system adaptation necessary for the definition of self-adaptive systems.

Arcaini et al. in [7] and Inglesia and Weyns in [58] aim to define self-adaptive systems by formally specifying the MAPE-K feedback loop. Arcaini et al. [7] show how MAPE-K loops can be explicitly formalised in terms of agents' actions using Abstract State Machines (ASM) transition rules to model the behaviour of self-adaptive systems. Concretely, the authors exploit the concept of multi-agent ASM to specify decentralised adaptation control by using MAPE computations. Although the authors aim at modelling and specifying self-adaptive systems, concretely the behavioural aspect of self-adaptation, their contribution primarily focuses on specifying the behaviour of the MAPE feedback loop (i. e., the managing system) and not the behaviour of the self-adaptive system as a whole. The other shortcoming is that the authors consider the adaptation as a result of the collaborative behaviour of multiple managing agents (i. e., MAPE-K loops). However, system adaptation is not necessarily an emerging property from collaboration, and its definition should be independent of the type and nature of the system. Finally, the authors consider the context and system in their formal specifications and informally the adaptation goals.

To support the design and the engineering of self-adaptive systems, Inglesia and Weyns in [58] derive a set of MAPE-K formal templates for designing feedback loops of self-adaptive systems. The proposed templates comprise: 1) behaviour specification templates for modelling different components of the MAPE-K loop and their interaction—using networks of timed automata (TA), and 2) and property specification templates for specifying required properties of the adaptive behaviour—based on timed computation tree logic (TCTL). Similar to the work of Arcaini et al. [7], the authors of [58] do not define the adaptive behaviour of the entire self-adaptive system but instead specify the MAPE behaviour, assuming that the MAPE behaviour will eventually adapt the managed system. As part of this work, the context, the system, and the adaptation goals are formally considered in the templates.

A more complete formalism has been proposed in a recent work by Bucchiarone and Mongiello [23], in which the authors introduce a formal framework to characterise different aspects of an ensemble-based software engineering. Concretely, they present 1) how to model dynamic software ensembles using Typed Graph Grammar (TGG), 2) how to specialise and re-configure ensembles, and 3) how to manage collective adaptations in an ensemble. As part of this work, the authors use TGGs combined with Labelled Transition Systems (LTSs)

to formally define system context, context-awareness, and system adaptation; however, only in the frame of system ensembles, which is the biggest shortcoming of this paper. However, adaptation as a system property should be considered and defined in independence from ensembles or system collaboration and not as an emerging property thereof. It is important to point out that compared to all the other analysed primary studies, there is a notable maturity in the work by Bucchiarone and Mongiello [23]. Concretely, this is the only work that defines system adaptation as part of their contribution formally. Furthermore, the authors also identify the importance of considering the context and the system, by explicitly considering the system functionality that adapts, as necessary aspects to discuss system adaptation and, therefore, self-adaptive systems.

Qureshi et al. in [103] take a different approach than the rest of the primary studies. In their work, the authors focus on defining the requirements for self-adaptive systems instead of defining self-adaptive systems. The authors tackle how the requirements problems (i. e., the problems solved during the requirements engineering) differ for self-adaptive systems compared to systems that are not self-adaptive. As it was previously observed, Broy et al. [20] and Bruni et al. [22] also tried to differentiate in their works how self-adaptive systems differ from those that are considered non-adaptive. The overarching objective of the work by Qureshi et al. in [103] is to identify concepts and relations that are necessary to be considered while eliciting and analysing requirements for self-adaptive systems. Therefore, the authors do not aim to define system adaptation, adaptive behaviour, nor MAPE behaviour as part of their work. Although this paper does not explicitly identify the relevance of the independent consideration of the system (i. e., the managed system that gains adaptation capabilities) as part of their formalism, this is the only paper in our primary studies that makes a distinction and formally considers the domain goals (referred to as mandatory goals as part of their work), and the adaptation goals (referred to as quality constraints).

Addressing RQ-A

Do the papers with formal definitions of self-adaptive systems also define system adaptation *as part of their contributions?* It is not possible to define self-adaptive systems without defining what it means for a system to adapt in the first place. However, our literature analysis showed that only one study formally defines system adaptation as part of their efforts to define self-adaptive systems; however, only in the frame of system ensembles. Two primary studies implicitly define system adaptation by specifying adaptive system behaviour as part of their contributions. And finally, two studies specify the MAPE behaviour (i. e., the behaviour of the managing system as part of a self-adaptive system), assuming that the MAPE behaviour will eventually adapt the managed system.

Addressing RQ-B

Which characteristics of the self-adaptive systems are considered in the existing formal definitions and specifications? If in RQ-A we focused on the behavioural aspect of self-adaptive systems, in RQ-B, we shift the focus to the structural aspects of these systems.

Concretely in this research question, we investigate which of the characteristics that have been recently consolidated in this field of research, as explained in Section 4.3.2, are considered in the existing body of work that formally defines self-adaptive systems. The most notable insight of our analysis is that *none* of the primary studies consider the aspect of uncertainty, both formally and informally, as part of their contribution. This is extremely surprising since the notion of uncertainty has been at the centre of the idea behind self-adaptive systems. Precisely the core motivation for self-adaptive systems is built on the unpredictable changes and *uncertainties* that trigger the need for system adaptation during the run-time of the system. This is also roughly how all the informal definitions available in the literature define self-adaptive systems, with a liberate use of the notion of uncertainties—a notion that is seemingly difficult to be put in formalism, as shown by our results. These results are another proof of the importance of having a clear, systematic, and formal definition of self-adaptive systems.

Almost all of the primary studies that we analysed consider the (states of the) context and system in some way as part of their formalism—the majority of them formally. This concludes that system adaptation and, therefore, self-adaptive systems cannot be defined in isolation from the context in which the self-adaptive systems operate and the properties of the system (i. e., the managed system) that gains the ability to adapt as part of a self-adaptive system.

Four out of nine primary studies (two formally and two informally) consider the concept of the adaptation goals, as we previously described them in Section 4.3.2, and identify that the system self-adapts in order to fulfil some quality objectives. However, the number of primary studies that consider the domain goals is much lower, and out of the nine primary studies only one study considers the domain goals. This is probably because this differentiation and the identification of the domain goals is much more subtle, but as we discussed in Section 4.3.2, it is necessary in order to argue when the system adapts and when does it simply function.

Addressing RQ-C

Which formal notations have been used across different works to define self-adaptive systems? Among the primary studies, three papers used Labelled Transition Systems (LTS)—in which one of them used Typed Graph Grammars (TGG) in combination with LTS. The rest of the studies used: Petri nets, FOCUS, Techne, Z language, abstract state machines (ASM), timed automata (TA) and timed computational tree logic (TCTL).

4.4 Discussion

4.4.1 Discussion on the results and future works

Despite the vibrant and growing community and the expanding interest in self-adaptive systems, our results have shown a sparsity of contributions that define self-adaptive systems formally. We derive various premises from the analysis and the results of our study, which set the foundation for the requirements for a holistic, formal definition.

The ideas of autonomic systems that introduced the MAPE-K conceptual model have profoundly impacted the engineering field and have initiated various new lines of research for the last two decades. Although MAPE-K gives some intuition behind the engineering of self-adaptive (and self-*) systems by the separation of concerns between the managing and the managed system, a more specific semantics of these two components is still missing. For instance, one can assume that every system that does some monitoring, planning, analysis, and execution and has some loose interpretation of the knowledge (e. g. every cyber-physical system), is self-adaptive by default. In response, the principles proposed by Weyns [116], concretely the internal principle that differentiates between the domain and the adaptation goals, have already made the initial steps in the direction of improving the terminology.

As we previously discussed in Section 4.3.2, it is paramount to distinguish between system functioning and system adapting. Making this distinction will set the foundation for defining *system adaptation* and, subsequently, self-adaptive systems. In our analysis, we observed that in three of the primary studies [20, 22, 103], the authors raised the question of the necessity to differ (self-)adaptive systems from the "ordinary", non-adaptive systems. However, the work by Bucchiarone and Mongiello [23] is the only study that contributes in this direction, in which the authors focus on identifying the system functionality that adapts; therefore, explicitly separating between system functioning and system adapting.

It is notable from the surveyed literature and our analysis that none of our primary studies (see Tab. 4.3) considers all the characteristics of self-adaptive systems as discussed in the principles [116]. The most unexpected insight from our results is that the notion of uncertainty has not been considered in the contributions of any of the primary studies, although uncertainty is considered the main reason for self-adaptive systems in the published papers on this topic and the informal definitions of these systems. So far, there is an intuitive understanding of the concept of uncertainty in self-adaptive systems, resulting in a clear need for more careful consideration of the aspect of uncertainty in this research domain. Concretely, how uncertainties can be represented, quantified and in general formalised as part of a formal definition of self-adaptive systems.

Finally, our results have shown that almost half of the primary studies provide their formalism by leveraging the aspects of collaboration [129], distribution [120], decentralisation [7], and ensembles [23] to define self-adaptive systems. However, system adaptation is not necessarily an emerging property from collaboration or decentralisation and should be defined in independence from these notions.

Based on our results and our findings, we can summarise that a potential formal definition of self-adaptive systems should provide a more precise semantics by 1) defining what it means for a system to adapt and how system adaptation differs from system function, 2) considering more systematically all the different characteristics of self-adaptive systems in its formalism, in particular the aspect of uncertainty, and 3) defining adaptation and self-adaptive system isolated from, e. g., collaboration and multi-agent systems.

4.4.2 Threats to validity

Although the systematic process for data collection and analysis followed the well-known accepted guidelines for systematic literature review [67, 68], there are some possible threats to validity that we summarise in the following.

Internal validity

In this study, we aimed to investigate how self-adaptive systems are defined in the literature. Finding this information in the papers we analysed was not always straightforward, especially while searching for informal definitions since this information was often implicitly included in the text. The expertise of the researchers also plays a role in this process; however, the potential bias of the researchers that conduct the systematic literature review is a common threat to validity. To mitigate this issue, voting was done by two of the authors. In case of conflicts, there was a follow-up discussion and a more in-depth paper analysis until a consensus was reached. On the other hand, searching for the formal definitions in the studies was much less complicated. Namely, in this case, we first searched if the analysed studies contained any formalism (which drastically reduced the search space). In case they did, we then proceeded with a thorough analysis of the paper, searching if the paper aims to define self-adaptive systems as part of their (formal) contributions. The voting on the formal definitions led to almost no conflicts among the authors.

External validity

Doing an automated search in six databases using the term "self-adaptive systems" yields hundreds of thousands of results. For that reason, we adopted the following two strategies, as previously explained in Section 4.2:

1. We implemented an iterative search process with pilot searches to define and fine-tune the search string to minimise the number of irrelevant studies. In each iteration, a subset of the collected data was manually inspected and analysed by two of the authors. The search string was refined based on the insights gained from the concrete iteration.
2. In our automated search, we either searched in the databases by meta-data (title, abstract, keywords) or only by title, depending on the advanced search options available in the concrete database. We assumed that if a paper defines self-adaptive systems, then that paper will certainly contain the word `(self-adapt*)` as part of these fields. However, there is the possibility to have missed some relevant studies by limiting the automated search in the databases only by meta-data.

However, not to compromise the completeness of the collected data, we analysed the complete initial pool of papers (1493 papers) and not only a random selection of these works. This proved to be the right decision, considering that the final set of primary studies contained only nine papers that could have been easily missed if we had decided to analyse only a random selection of the initial pool of papers.

Reliability

To ensure that our research findings can be replicated, as part of this paper, we have made available a reproducible package with the selected studies in each of the phases of the methodology. The package contains all the necessary data for replication, including the final queries that we used for the automated search in the databases and the authors' votes. To mitigate the inherent bias that each researcher has due to their background and experience, we have ensured that multiple researchers made the paper selection and the data extraction and analysis. Precisely, during the analysis in Phases 4 and 6 of our methodology, we introduced a voting process in which, if the authors classify a paper differently, a discussion took place until the voters have reached a unified decision about the study. On the other hand, the reliability of the used databases and the replication of the automated search with the specific queries is something that we cannot account for.

4.5 Related Work

To the best of our knowledge, this is the first study that has focused on systematically collecting and analysing how self-adaptive systems are defined. Although the interest in self-adaptive systems has been rapidly growing, the concrete semantics of the core terminology is still missing. Namely, the literature still lacks a consensus on a definition—understanding why this is the case and getting a better overview of the existing body of literature was the motivating factor for our study.

Many other systematic reviews and mapping studies with different objectives were conducted over the years in the literature. However, they were all focusing on other aspects related to self-adaptive systems, for example, engineering approaches for self-adaptive systems [73, 74, 122, 119], the use of formal methods in self-adaptive systems [121], and two recent works in which the authors focused on decentralisation in self-adaptation [101] and the application of machine learning in self-adaptive systems [48]. Besides the existing systematic literature reviews and mapping studies, there are a couple of other surveys and roadmaps on future research challenges [80, 84, 107, 118]. In contrast to these works, in our systematic literature review, we aim to consolidate the existing (formal) definitions of self-adaptive systems and, more importantly, understand their limitations, which sets the foundation for a future establishment of a more unified understanding of these systems. An improved terminology semantics will complement the existing works in this field, including the contributions from the other systematic reviews, mapping studies, and roadmaps presented above.

Motivated by similar incentives as our study, only putting the focus on self-awareness instead of self-adaptation, Elhabbash et al. in [37] have conducted a systematic literature review on the usage of self-awareness in software engineering. Among other objectives, the authors also summarise and analyse how self-aware systems have been defined in the literature. Please note that in this study, the authors only focus on summarising the informal definitions of these systems. Although most of the researchers in the literature use the terms self-adaptation and self-awareness interchangeably, there are some prior works [71, 81, 94], in which the authors distinguish these terms and consider self-awareness

as an "enabler" or a precondition for self-adaptive systems. In the future, if we have a clearer and more precise definition and understanding of self-adaptive systems, this will also help us to better distinguish self-adaptive systems from other self-* systems, such as self-aware systems.

4.6 Conclusion

The lack of commonly accepted definitions and the ambiguous description of the terminology adds complexity to an already complex field of research. In response, our main objective in this paper was to get better insights and summarise the existing body of literature by systematically reviewing how self-adaptive systems have been defined in the prior works in this field. First and foremost, we wanted to investigate if the existing works define self-adaptive systems and how many of these works provide any definition, putting a special emphasis and qualitatively analysing the formal definitions of these systems later in the paper. Our results showed that 1) the majority of the papers we analysed do not even provide an informal definition or description of what they consider a self-adaptive system to be, and 2) only nine papers aimed to define or specify self-adaptive systems in their contributions formally. The results of our study clearly revealed that the problem of defining self-adaptive systems remains under-researched, which to some extent, explains why the research field lacks more unified definitions—or generally a better understanding of the terminology. We hope that with our study, we raise awareness on this important matter and show the great potential for future research that will focus on creating a more precise understanding and, ideally, formal definitions of self-adaptive systems that will be accepted on a broader range.

Our systematic literature review also provides a foundation for improving the terminology. Concretely, through our qualitative analysis of the existing formal definitions of self-adaptive systems, we also summarised their limitations and shortcoming, based on which we elicit requirements for a potential holistic and unified formal definition of these systems. Increasing the clarity of the terminology will support and complement the already existing approaches and methods for engineering self-adaptive systems and also open new directions of research in the future, enabling the community to endeavour to a fuller extent.

5 Defining adaptivity and logical architecture for engineering (smart) self-adaptive cyber-physical systems

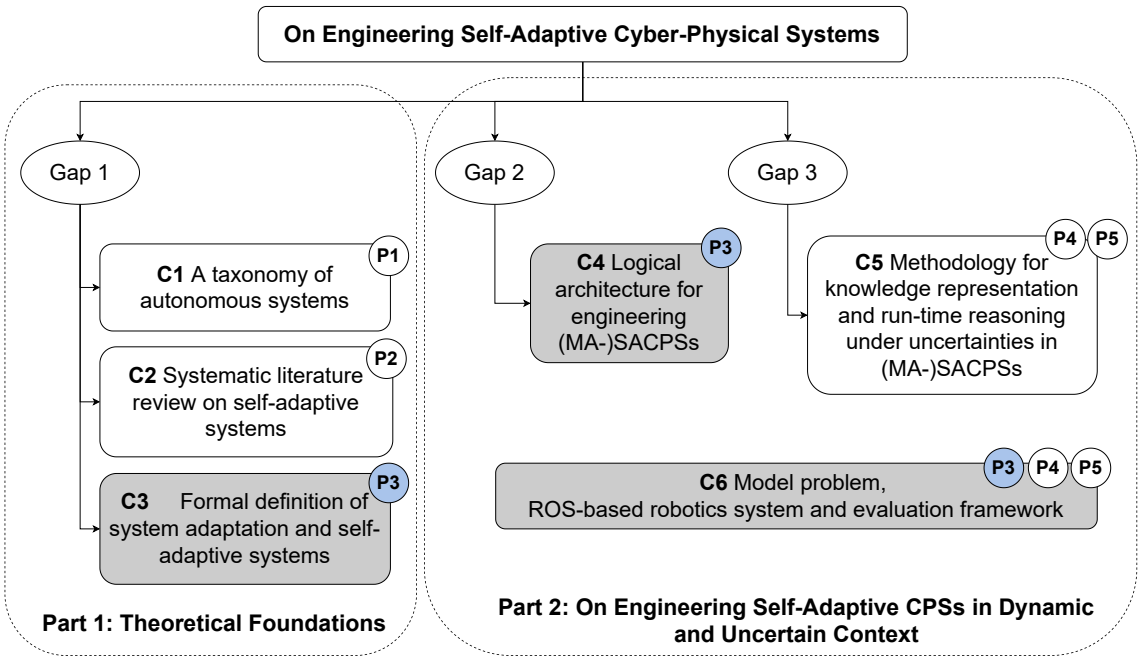


Figure 5.1: *Mapping of Gaps, Contributions and Publications to the structure of this thesis.*

Summary: In our systematic literature review from Chapter 4, we provided an overview of the current state-of-the-art on how self-adaptive systems are defined in the literature so far. Furthermore, we discussed the limitations and shortcomings of the existing formal definitions, based on which we elicit requirements for a potential holistic and unified formal definition of these systems. In Chapter 4, we also discussed that defining the property of *system adaptation* is the first step toward defining self-adaptive systems. In response, as part of this chapter, we first formally define system adaptation, and based on our definition, we provide a more precise characterization and semantics of self-adaptive systems. However, definitions are generally declarative and descriptive, and they do not provide constructive insights into how to engineer self-adaptive systems.

Additionally, as discussed in Section 1.1, the proposed MAPE-K model is a de facto conceptual model for engineering not only self-adaptive systems but self-* systems in general. As a result, this conceptual model does not help in differentiating how the engineering processes for building a self-adaptive system differ from the engineering of other self-* systems. Therefore, as a second contribution of this chapter, we present a

logical architecture for engineering self-adaptive CPSs that narrows the gap between: 1) the MAPE-K conceptual architecture and 2) a potential technical implementation for a class of self-adaptive CPSs that operate in dynamic, uncertain and partially observable context.

Problem: The problem that we tackle in this paper is two-fold: Despite the notable advancements on many fronts, engineering self-adaptive systems remains more challenging than traditional systems [120], especially when the concepts of self-adaptation are aligned with the new emerging technologies. Although there are numerous technical challenges, the lack of a clear definition of the core terminology and a shared understanding of self-adaptive systems might be what hinders the progress the most. According to Weyns, self-adaptive systems have not been defined yet [116], and the lack of clear definitions of the terminology is the most prominent contemporary challenge in the field of engineering of self-adaptive systems [118, 119]. Different models and frameworks on engineering self-adaptive systems [44, 46, 38, 70] have been proposed in the literature, and although these works achieved distinguished success, defining self-adaptive systems was not their aim. There have been a few past efforts to formally define self-adaptive systems [20, 120, 22, 23]; however, none of these formalizations is accepted and used by the community as a reference definition.

Broy [17], Lints [82], and Weyns [116] have all independently observed that across the literature, the terms system adaptation and self-adaptive systems are mainly used intuitively without a deeper understanding of their meaning. 1) The intuitive interpretation of the notion of *system adaptation*, which primarily relies on the spoken language, and 2) the informal definitions of self-adaptive systems are fairly ambiguous and result in underspecified usage of these terms. Although such usage might suffice in some instances, this is not the case in engineering and science, where if and when a system behaves adaptively cannot be answered through intuition, and a more rigorous definition is necessary [82]. The ambiguity in the understanding of system adaptation results in ambiguity in the understanding of *what* self-adaptive systems are. This is the first problem that we tackle as part of this paper.

The lack of definitions and terminological clarity has different software engineering consequences and implications, for instance, *how* to build or engineer these systems that go beyond the famous MAPE-K conceptual model. Establishing a more precise vocabulary, specifications, and definitions of self-adaptive systems, or in general, improving the foundational clarity of the terminology, is the first step towards the modeling, design, implementation, and evaluation of these systems. However, definitions remain mainly descriptive, as they do not provide constructive insights into how self-adaptive systems are designed and built. As a result, a need emerges for architectural solutions—beyond the conceptual MAPE-K model—on *how* to build self-adaptive systems, which is the second problem that we tackle in the second half of this paper.

Gap: Despite the extensive work on self-adaptive systems, software and system engineering literature still needs a precise, comprehensive, applicable, and broadly accepted (formal) definition of self-adaptive systems. To define self-adaptive systems and understand self-adaptation, we first need to understand what the concept of *system adaptation* means.

However, we can observe that most of the available works in this domain focus exclusively on self-adaptive systems without first clarifying and defining what is understood under the notion of *system adaptation*, which is an essential prerequisite for a subsequent definition of self-adaptive systems. This furthermore results in a lack of foundational clarity of self-adaptive systems, including 1) a lack of clarity on the exact processes of adaptation and self-adaptation, 2) the minimal requirements for a system to be self-adaptive and, in general, 3) how self-adaptive systems differ from the "ordinary," non-adaptive systems. Defining adaptation as a system property is the core pillar for defining self-adaptive systems, and is necessary to compare the existing and future works in this field.

Probably due to the scarcity of the precise definition of the terminology, there is also a deficiency of architectures that can serve as a blueprint for engineering self-adaptive systems. To the best of our knowledge, no prior work provides concrete architectures for engineering decentralized and autonomous self-adaptive CPSs that operate in dynamic, changing, and uncertain contexts that are only partially observable by the CPSs. Since CPSs are dynamic themselves and operate in a dynamic context, having knowledge in the adaptation logic that is hard-coded during the design of the self-adaptive system does not suffice. Instead, the adaptation logic of dynamic self-adaptive systems should have mechanisms that will enable it to be continuously updated during the system's run-time to reflect the context's run-time states and the system that are relevant for the adaptation.

Method/Solution: In this paper, based on the elicited requirements from Chapter 4, we *first* formally define system adaptation and show how i) the quality aspects and the separation between business and adaptation goals and ii) the distinction between system functioning and system adapting play a central role in defining (self-)adaptive systems. *Second*, within our formal framework, we provide a structure for engineering self-adaptive CPSs that narrows the gap between the formal definitions and potential technical implementations for a class of systems. Concretely, we propose a logical architecture that can serve as guidance or a blueprint for engineering decentralized and autonomous self-adaptive CPSs that operate in changing, uncertain, and partially observable contexts. In our logical architecture, the adaptation logic "learns" and changes itself during run-time to accommodate run-time uncertainties and to continue reflecting the relevant aspects from the states of the context and the system for the concrete adaptation.

Findings: In order to debate if a system is adaptive, we first need to specify the right framing according to which the system is considered to be adaptive:

- What is the system function (sf) that is considered as adaptive?
- According to which adaptation goals does the system (i. e., the system function (sf)) adapt?
- According to which context or system conditions (changes, uncertainties) does the system (i. e., the system function sf)) adapt?
- What is the time period in which the system is considered adaptive?
- And lastly, emerging from our formal definition of system adaptation: under which convergence parameters ℓ and ε is the system considered as adaptive?

Our proposed framing needs to be considered in the engineering processes of every self-adaptive system, and it should conveniently drive the design, implementation, and evaluation of these systems. We further exemplify the usage of the in framing Appendix A.

Results: The conducted evaluation as part of this paper enabled us to show the validity of the proposed formal definition of system adaptation and the proposed framing for engineering (self-)adaptive systems. We summarize the steps we conducted to bridge the gap from our proposed formal definition to the empirical results as independent contributions in the following.

Contribution: As part of this paper, we formally define system adaptation, propose the necessary framing for engineering (self-)adaptive systems and provide a preliminary characterization of self-adaptive systems. The differentiation between the nominal system function (sf) and system adaptation enables us to differentiate between ordinary and adaptive systems. This is further supported by the separation between the business and adaptation goals. The business goals are related to the functional requirements of the systems, i. e., the system function (sf) of the managed system—the system that gains adaptation capabilities as part of a self-adaptive system. However, besides attaining the business goals, the main reason for system adaptation is attaining one or more adaptation goals despite the dynamic and uncertain internal and contextual conditions. If the business goals are related to the system's functional requirements (i. e., the system function), then the adaptation goals are related to some non-functional requirements or quality objectives.

To quantify and measure the system adaptation, we introduce a metric referred to as *Quality Function*. Concretely, the Quality Function measures the degree of fulfillment of the adaptation (and the business) goals over time. Creating and defining the Quality Function—which includes specification and quantification of the business and the adaptation goals—is a complex, multi-stage process that is use case-specific and is left to the engineers of the self-adaptive systems. However, to validate and evaluate our formal and theoretical contributions, including the Quality Function that is essential for measuring system adaptation and a significant part of our logical architecture, we also propose a calculation of the Quality Function, which is yet another contribution of this paper. To achieve this, we 1) designed a model problem from the robotic domain, based on which we 2) implemented a simulated, ROS-based, and multi-agent system, and 3) implemented a so-called *evaluation framework* tooling. These are three additional technical contributions as part of this work. The evaluation framework can be used in any self-adaptive system that adopts the standardized interface of the framework. The calculation of the Quality Function in our evaluation framework is applicable and re-usable for a class of systems from the CPSs domain, and with minor modifications, it could be universally usable for measuring system adaptation for every self-adaptive system.

Within the proposed theoretical concepts and formalisms from the first part of this paper, in the second part of this paper, we propose a logical architecture for engineering self-adaptive CPSs that operate in a dynamic and changing context, from which the CPSs make partial and uncertain observations. Concretely, our logical architecture enables the

self-adaptive CPS to change its adaptation logic (i. e., the knowledge in the adaptation logic) to reflect the actual state of the context during run-time, which is something that none of the previously proposed reference architectures [3, 14] for engineering self-adaptive systems support. Additionally, the logical architecture defines how different components interact and what information they exchange. This provides another dimension of semantic precision, which has not been considered in the previous works [66, 44, 91, 123, 101]. Finally, as part of this paper, we discuss open research problems and future research directions stemming from the proposed formalization of system adaptation and the logical architecture.

Limitations: There are a couple of limitations that we summarize in the following.

Limitations of the formal definition of system adaptation. As part of the paper, we define system adaptation as a sequence of functions $sf_0, sf_1, \ldots, sf_m, \ldots, sf_n, \ldots$ at time $t = 0, 1, \ldots, m, \ldots, n, \ldots$ that improves over time. These functions improve the fulfillment of the adaptation goals (and the business goals) at run-time by mapping a function sf_m to another "better" function sf_n, where $n > m$,. However, our formulation could be interpreted in two different ways.

According to the first interpretation, this is a sequence of the system function sf adapted or improved by the adaptation logic at different points in time, with improved Q over time. In other words, the system adaptation is a sequence of functions sf_0, sf_1, $\ldots, sf_m, \ldots, sf_n, \ldots$ at time $t = 0, 1, \ldots, m, \ldots, n, \ldots$, with $n > m$, that are assessed (according to Q) and *their assessment* improves over time (e. g., sf_n is "better" than sf_m) despite the changing and uncertain run-time conditions. How Q is defined, specified and measured (i. e., assessed) is left to the engineers of the self-adaptive system. This includes how Q measures the system function sf at specific point in time. As part of the work presented in this paper, we assume that the value of the system function sf is measured through the value of the states of the context and the system (i. e., the managed element). In other words, the calculation of Q is an approximation, in which we abstract the input and the outputs of sf, since we assume that they are explicitly encoded in the states. In a nutshell, we can say that the sf of an "ordinary", non-adaptive system system has a sequence of system states $\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_m, \ldots, \sigma_n, \ldots$ and this sequence is improved in the case of a self-adaptive system. Concretely, in case of a self-adaptive system, sf is changed by the adaptation logic over time $t = 0, 1, \ldots, m, \ldots, n, \ldots$, resulting in a different sequence of states $\sigma_0, \sigma'_1, \sigma'_2, \ldots, \sigma'_m, \ldots, \sigma'_n, \ldots$, compared to a non-adaptive system.

According to the second interpretation, this is a sequence of *different functions* that *evolve* over time. For a function to change, it either changes its input or output interface or changes the state transition by itself during run-time. In the first interpretation, we talk about system adaptation, whereas in the second interpretation, we talk about system evolution, which is part of a completely other research domain and is out of the scope of this dissertation.

Lastly, as part of this paper in this chapter, we only formalize system adaptation and provide a preliminary characterization of self-adaptive systems. Based on the definition of system adaptation from this paper, in Appendix A, we:

1) we specify the process of self-adaptation,
2) extended the preliminary characterization and minimum requirements for self-adaptive systems, and lastly,
3) formally define (passive and active) self-adaptive systems.

Limitations of the calculation of the Quality Function. For calculating the Quality Function, we assumed that the state of the context and the systems are available. However, this is not always possible, depending on the concrete system under consideration. For example, due to the partiality in the observations of the CPSs that we focus on as part of this thesis, these systems do not have access to the complete context state relevant for system adaptation. Nonetheless, in the case of some software systems, these limitations might not exist, and this is something that emerges only from the complexity of the system or the model problem that we considered as part of this thesis. As part of this paper, we propose two solutions to this limitation; however, this research problem still remains open for additional research contributions in the future.

Limitations of the implementation. The Quality Function can be used for two purposes: i) to define the system adaptation, and ii) to measure its (approximate) value at run-time (see the previous limitation). Furthermore, the measured, approximated value can be used i) to (passively) evaluate the system adaptation (as we do in the validation of the formal definitions in this paper) and ii) to (actively) steer the self-adaptation and decide which adaptation actions shall be triggered based on the value of Q. Based on this, in Appendix A, while defining self-adaptive systems, we differentiate between passive and active (also referred to as true) self-adaptation. Please note that the theory of active self-adaptation, in which the value of Q is used to actively steer the self-adaptation, is not realized as part of the implementation of the use case in this thesis. This is because some of our theoretical findings, which we reached towards the end of this dissertation, opened various research and technical problems that remained out of the scope of the implementation. However, it is important to emphasize that this is not a limitation of the logical architecture that we propose as part of this paper (since we have dedicated components and processes as part of our architecture); instead, as previously said, it is a limitation of the implementation of the system.

Author Contribution: A. Petrovska and A. Pretschner conceived and discussed the problem statement. The initial drafts of the theoretical contributions and formalisms were developed by A. Petrovska and A. Pretschner, and were later discussed and refined with the help from S. Kugele and T. Hutzelemann. A. Petrovska and T. Beffart developed the initial version of logical architecture, which was later reviewed and refined in close discussion with A. Pretschner and S. Kugele. The model problem from the robotics system, including the implementation of the robotics system, was done by A. Petrovska. The implementation of the evaluation framework, including the Quality Function, was done by S. Bergemann and A. Petrovska. S. Bergemann and A. Petrovska also designed the experiments and jointly analysed the results. S. Bergemann conducted the experiments. The paper was written by A. Petrovska and revised by A. Pretschner, S. Kugele and T. Hutzelemann.

Information and Software Technology 147 (2022) 106866

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Defining adaptivity and logical architecture for engineering (smart) self-adaptive cyber–physical systems

Ana Petrovska [a,*], Stefan Kugele [b], Thomas Hutzelmann [a], Theo Beffart [a], Sebastian Bergemann [a], Alexander Pretschner [a]

[a] Department of Informatics, Technische Universität München, Boltzmannstasse 3, Garching bei München, Germany
[b] Research Institute AImotion Bavaria, Technische Hochschule Ingolstadt, Esplanade 10, 85049 Ingolstadt, Germany

ARTICLE INFO

Keywords:
Adaptivity
Quality function
Knowledge
Logical architecture
Self-adaptive cyber–physical systems

ABSTRACT

Context: Modern cyber–physical systems (CPSs) are embedded in the physical world and intrinsically operate in a continuously changing and uncertain environment or *operational context*. To meet their business goals and preserve or even improve specific adaptation goals, besides the variety of run-time uncertainties and changes to which the CPSs are exposed—the systems need to self-adapt.
Objective: The current literature in this domain still lacks a precise definition of what self-adaptive systems are and how they differ from those considered non-adaptive. Therefore, in order to answer *how* to engineer self-adaptive CPSs or self-adaptive systems in general, we first need to answer *what* is adaptivity, correspondingly self-adaptive systems.
Method: In this paper, we first formally define the notion of adaptivity. Second, within the frame of the formal definitions, we propose a logical architecture for engineering decentralised self-adaptive CPSs that operate in dynamic, uncertain, and partially observable operational contexts. This logical architecture provides a structure and serves as a foundation for the implementation of a class of self-adaptive CPSs.
Results: First, our results show that in order to answer if a system is adaptive, the right framing is necessary: the system's adaptation goals, its context, and the time period in which the system is adaptive. Second, we discuss the benefits of our architecture by comparing it with the MAPE-K conceptual model.
Conclusion: Commonly accepted definitions of adaptivity and self-adaptive systems are necessary for work in this domain to be compared and discussed since the same terms are often used with different semantics. Furthermore, in modern self-adaptive CPSs, which operate in dynamic and uncertain contexts, it is insufficient if the adaptation logic is specified during the system's design, but instead, the adaptation logic itself needs to adapt and "learn" during run-time.

1. Introduction

In recent years, the widespread availability of cost-effective embedded systems with increasing computation power and the expansion of wireless networks has led to the rise of Cyber–Physical Systems (CPSs) in many different domains. CPSs are software-intensive systems that are embedded in the physical world. They monitor, control, and coordinate processes in both the physical and the digital world. These modern CPSs are often mobile, decentralised, and operate autonomously, e.g., robots, drones and self-driving cars. Additionally, multiple CPSs can communicate and work collaboratively towards achieving one or several common objectives, for instance, by forming fleets of robots or drones, or platoons of automated self-driving cars. Multiple collaborating CPSs can provide shared and more complex functionalities that a single system in isolation cannot attain. As a consequence of their interaction and deployment into the physical world, CPSs operate in a dynamic and uncertain environment, or *execution context* [1–4]. The context and the CPSs themselves are continuously changing in unanticipated ways that could not be known during the systems' design. Despite the changes and uncertainties at run-time, business continuity of modern CPSs requires the systems to resume operating efficiently and reliably within inherently dynamic conditions. Ramirez et al. [5] classify the sources of run-time uncertainties in adaptive systems in two groups: (1) *internal* uncertainties that originate from the self-adaptive system itself, i.e., sensor failure, sensor imprecision, sensor noise and effectors,

* Corresponding author.
E-mail addresses: ana.petrovska@tum.de (A. Petrovska), stefan.kugele@thi.de (S. Kugele), t.hutzelmann@tum.de (T. Hutzelmann), theo.beffart@tum.de (T. Beffart), sebastian.bergemann@tum.de (S. Bergemann), alexander.pretschner@tum.de (A. Pretschner).

https://doi.org/10.1016/j.infsof.2022.106866
Received 1 May 2021; Received in revised form 14 January 2022; Accepted 28 January 2022
Available online 19 February 2022

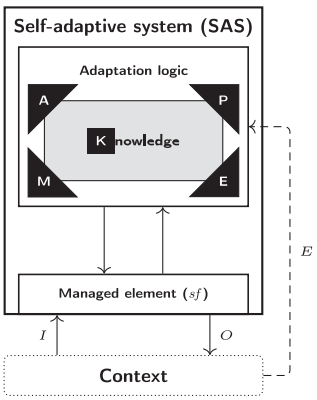


Fig. 1. Updated conceptual model of a self-adaptive system from [3].

and (2) *external*, e.g., unpredictable environment. Additionally, due to the sensing limitations of the hardware (i.e., the limited sensor range) of the CPSs, they observe the environment or the context in which they operate only partially. Concretely, the CPSs cannot observe the complete state of their context during their operation, introducing another dimension of run-time uncertainty. A common approach to deal with run-time changes and uncertainties that could not be fully anticipated during the design of the CPSs—while preserving the systems' performance or certain quality objectives—is to make the CPSs self-adaptive.

In architecture-based self-adaptation [4,6–8], a common approach to introduce the necessary self-adaptation capabilities is to extend the CPS, which is the *managed element* as part of a self-adaptive system, with an *adaptation logic* (see Fig. 1). Traditionally, the adaptation logic is based on the MAPE—a closed feedback loop [6]. In previous works, the adaptation logic has also been referred to as managing system [3], or as autonomic manager [6]. The MAPE-K loop is comprised of four consecutive functions or phases: *Monitor*, *Analysis*, *Plan*, and *Execute*, with *Knowledge* that is shared across all four stages of the loop.

Problem: Despite the notable advancements on many fronts, engineering (self-)adaptive systems remains more challenging than traditional systems [9], especially when the concepts of self-adaptivity are aligned with new emerging technologies found in CPSs. The difficulty in defining self-adaptive systems primarily originates from the lack of understanding (1) what is a system's adaptivity, and (2) how self-adaptive systems differ from *ordinary* systems that are considered non-adaptive.

As part of this paper, we explore the following hypothesis:

> **Hypothesis:** To argue if a system is adaptive or not, we first need to specify its framing:
>
> 1. The adaptation goals (i.e., one or more quality objectives) according to which the system is considered to behave adaptively,
> 2. The context and the system situations (i.e., different changes and uncertainties) in which the system adapts, and
> 3. The time frame in which the system is considered as adaptive.

Due to the scarcity of precise definitions of adaptivity and the lack of a precise characterisation of self-adaptive systems, there is a deficiency of architectures and design patterns that can serve as a blueprint for engineering these systems. Establishing definitions and understanding what self-adaptive systems are, is the first step towards specifying, modelling and designing these systems. However, definitions still remain mainly descriptive, as definitions do not provide constructive insights into how self-adaptive systems are designed and built. Therefore, the identified problem is two-fold: (1) there is a gap between the MAPE-K (see Fig. 1) conceptualisation and potential technical implementations of a self-adaptive system. Concretely, the conceptual MAPE-K model has a very high level of abstraction and is not particularly helpful in designing and implementing the actual system; and (2) the conceptual MAPE-K model does not provide concrete insights on how self-adaptive systems (engineered according to MAPE-K) differ from "ordinary" systems that are non-adaptive. As a result, a need emerges for architectural solutions on *how* to build self-adaptive systems at some intermediate level of abstraction—beyond the conceptual MAPE-K model—especially when SACPSs that operate in changing, uncertain and partially observable contexts are in focus.

Gap: Existing works do not provide concrete architectures for engineering self-adaptive systems, particularly decentralised and autonomous self-adaptive CPS (SACPS) that operate in changing and uncertain contexts that are only partially observable by the CPSs. Until now, in the literature, there are several proposed design patterns for self-adaptive systems [10,11]; however, these works mainly focus on different combinations in the decentralisation of the four phases of the MAPE, and they *exclude the knowledge component*. Although different MAPE-based patterns are more informative regarding the system's design, inherently they have the same limitations as the MAPE-K closed feedback loop itself: their high level of abstraction and low level of details, which does not provide any characterisation of how a system built upon the MAPE loop differentiates from the *ordinary*, non-adaptive systems.

Solution: As part of this work, *first* we formally define context and adaptivity and show how quality plays a central role in understanding and defining self-adaptive systems. *Second*, we embed the formal concepts of adaptivity into a logical architecture for engineering SACPSs that narrows the gap between the formal foundations and potential technical implementations of a class of use cases. Concretely, the proposed logical architecture can serve as guidance or a blueprint for engineering autonomous and decentralised SACPSs. In our logical architecture, not only the managed elements (i.e., the CPSs) but the adaptation logic adapts as well. In other words, the adaptation logic "learns" and changes itself during run-time to accommodate run-time changes and uncertainties that could not be anticipated during the design of the self-adaptive system. The proposed formal definitions of adaptivity can be considered as orthogonal to architectures at different levels of abstraction: (1) the MAPE-K conceptual model at the highest level of abstraction and the lowest level of details, (2) the proposed logical architecture at an intermediate level of abstraction and intermediate level of details, and (3) a potential technical architecture of an implementation—instantiated from our logical architecture—at the lowest level of abstraction and highest level of details.

Structure: The remainder of the paper is structured as follows: in Section 2, we present the use case, followed by related work in Section 3. In Section 4, we define adaptivity before giving a high-level overview of the logical architecture in Section 5. In Section 6, we explain in more detail the design-time components, and in Section 7, we explain the run-time components of the architecture. In Section 8, we evaluate the proposed formal notions of adaptivity on an implementation from the robotics domain and discuss the benefit of our logical architecture by comparing it to prior works. In our discussion in Section 9, we discuss the threats to validity, the implications from the technical implementation of systems according to our logical architecture and future research directions. Section 10 concludes the paper.

2. General setting and use case description

2.1. Class of use cases from the CPSs domain and general setting

The logical architecture proposed in this paper can be used as a blueprint for the development of the technical architecture, i.e.,

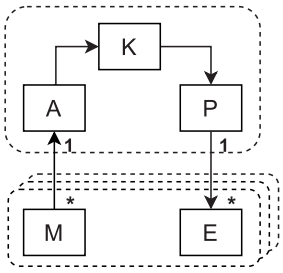


Fig. 2. Master–Slave pattern of the MAPE-K loop.
Source: Updated from [10].

the implementation for the class of CPSs explained in this section. Concretely, we consider every use case in which one or more mobile and decentralised CPSs autonomously traverse an environment to discover and collaboratively attain tasks that are continuously appearing. What we aim at adapting in this class of use cases is how the CPSs traverse the dynamically changing and uncertain context in which they operate, despite the partial and uncertain observations that the CPSs make during run-time. Autonomous, in this case, refers to the CPSs' capabilities to navigate and move in space without the need for physical or electro-mechanical control from a guidance device. The CPSs could operate in two-dimensional (e.g., robots, self-driving cars) or three-dimensional (e.g., drones, UAVs) environments or contexts. Although the properties of mobility and autonomy are important for the targeted class of systems, our architecture can guide the engineering of self-adaptive systems for all the CPS which are (1) *decentralised* with (2) *uncertain and partial monitoring or observation capabilities* of the dynamic and the uncertain context in which the systems operate. This also includes, for example, stationary agents (e.g., radio antennas, different IoT devices, etc.) that comply with the properties mentioned above. In these cases, the majority of the components from the architecture can be re-used except for the logic of the Planning Logic component (see Fig. 4).

Self-adaptive systems have two types of goals: *business goals* related to the functional requirements of the system, and *adaptation goals* related to the non-functional requirements or different quality objectives of the system [2]. In our class of use cases, the requirements for CPSs are to fulfil their function (i.e., the business goals) by discovering and attaining tasks. The CPSs complete the tasks by navigating and reaching their locations. In addition to fulfilling their business goals, every self-adaptive system, including SACPS, also needs to fulfil one or more adaptation goals, which are different quality objectives. The adaptation goals of the SACPSs need to be satisfied despite different run-time uncertainties and changes that could not be anticipated during the design of the systems (see Section 1).

2.2. Requirements of the SACPSs

In this work, we assume that every decentralised CPS is a managed element as part of the SACPS and contains monitoring and execution capabilities, i.e., it can perform the M and E phases of the MAPE-K independently on a local level. Namely, each CPS is able to produce its own observations from the context and execute its respective planned actions independently. Concretely, the CPSs monitor and can only detect appearing tasks within their range of observation, which is partial (covering only part of their context) and uncertain. Furthermore, by "executing their respective planned actions independently", we refer to the ability of each CPS to autonomously localise itself and navigate to the locations of its tasks, once the tasks—concretely the location of the tasks—are assigned to the CPS by the adaptation logic.

The other three phases of the MAPE-K: analysis, planning, and knowledge are centralised in the adaptation logic and shared among all the CPSs. Note that although these components are centralised as part of the logical architecture, they could be realised in a distributed manner in the technical implementation of the system. The adaptation logic needs to (1) consider all the uncertain and partial context observations from every CPSs, (2) consolidate that information and resolve all the conflicts that could potentially originate from the uncertain and partial observations from the previous step, (3) update the knowledge in the adaptation logic, based on the consolidated and aggregated information from the decentralised observations, (4) and plan the optimal actions for all the CPSs (based on the adapted knowledge), according to the adaptation goals of the overall SACPS. In sum, we propose an architectural solution for SACPSs in which the MAPE-K loops are structured according to the Master–Slave pattern [10], shown in Fig. 2,[1] and satisfy all the properties described above. The managed elements (the CPSs) fulfil the functionality of the SACPS by detecting and navigating to the locations of the tasks, and the behaviour of the CPSs (which task to pursue next) is adapted by the adaptation logic according to specific adaptation goals.

2.3. Robotics running example

From the class of use cases, we instantiate a specific use case from the robotics domain. The robotics use case is also used as a running example to demonstrate the usage of the logical architecture throughout the rest of the paper. In our example, multiple robots clean a room in which dirt patches are continuously spawned with unknown location patterns and frequencies. Each robot can autonomously move to a destination—usually, a task's location while avoiding static obstacles (e.g., walls) or dynamic obstacles (e.g., other robots, humans) along its way. Additionally, sometimes, the robots observe tasks on locations where there are no tasks, or fail to observe an existing task. The dirt patches represent the cleaning tasks for the robots. Thus, the robots fulfil their business goal by detecting and cleaning the dirt tasks. In addition to simply keeping the room clean, we also want to improve *the quality of this cleaning process* (i.e., the adaptation goal). The robots need to fulfil both: the business and the adaptation goals, despite the existence of various external and internal uncertainties. The system function that we want to adapt in our running example is the global task assignment for all the robots, which gets adapted based on the "best", most complete representation of the dynamic, uncertain and partially observable context that is learned and aggregated over time, in order to maximise the specific adaptation goals despite various run-time uncertainties.

The robots explore the room and discover new tasks in a distributed manner with a scanner capable of sensing 360 degrees radius, in our example, a LiDAR sensor. However, due to their internal technical limitations, the robots are subjected to different sensor run-time uncertainties and partiality in the observations. The internal systems' uncertainties might originate from different sensor uncertainties, e.g., sensor noise, ambiguity, imprecision and even a complete sensor failure [5]. Furthermore, due to their sensors' technical limitations (i.e., limited sensing range), the CPSs produce only partial observations of the context in which they operate, introducing another dimension of internal uncertainty. Due to the different sensor uncertainties, a divergence could occur between the measured value (e.g., the measured location of the task) and the real value (e.g., the actual location of the task), which could lead to two or more values of the same property disagreeing with each other (e.g., two or more robots observing the same task but at different locations). This results in the robots not knowing the actual context (i.e., the room) in which they operate in regard to the task appearance, which affects the fulfilment of the adaptation goals.

[1] (1-*) is one-to-many cardinality.

3. Related work

In this section, we examine the most relevant approaches with respect to our proposed logical architecture.

Models. More than twenty years ago, Kephart and Chess, in their famous IBM manifesto on autonomic computing [6] have proposed the MAPE-K *conceptual model*. As explained previously in Section 1, most of the architecture-based self-adaptation systems are built upon the MAPE-K loop, or at least on different variations of the five phases of the MAPE-K. However, the MAPE-K is a conceptual model at a high level of abstraction with little detail and is not necessarily helpful in the design of an actual technical implementation of a self-adaptive system. Furthermore, one might argue that there is no clear distinction of how self-adaptive systems built according to the MAPE-K differ from the *ordinary*, non-adaptive systems, since every system to some degree incorporates notions of monitoring, analysis, planning and execution.

Other software models have also been used for the development of different parts of self-adaptive software systems. Zhang and Cheng [12] propose a model-driven software development process for dynamically adaptive programs, focusing on behavioural modelling. Concretely, their paper introduces an approach to create formal models for the behaviour of adaptive programs. Weyns et al. [9] have also focused on using models as formal specifications of self-adaptive software systems. The authors propose FORMS, which enables precise descriptions of architectural characteristics in the early design of the systems. Although both papers provide specifications for engineering self-adaptive software systems, they do not define what adaptivity is, nor how systems with adaptive behaviour differ from systems that are considered non-adaptive. Additionally, the formal specifications of their models do not consider the notion of context and quality (adaptation goals), which to the best of our understanding, are necessary to define any concept related to adaptivity (further explained in Section 4).

Patterns. Weyns et al. [10] propose patterns for decentralised MAPE control loops in self-adaptive systems. The authors argue that when systems are large, complex and heterogeneous, a single MAPE loop might be insufficient for managing all the system's adaptations, and the need to decentralise the MAPE phases emerges. In [10], the authors develop a systematic approach for describing multiple interacting MAPE loops, based on which they propose five patterns for MAPE-based decentralised control in self-adaptive systems. Although the different patterns for decentralisation of the MAPE loops can foster more structured formations for engineering self-adaptive systems, we see the same concern as with the MAPE-K loop itself—they remain on a very high level of abstraction, and they are not necessarily beneficial for the design of an actual physical architecture of a self-adaptive system.

Musil et al. [13] report the results of a systematic survey on CPSs that combine different self-adaptation mechanisms across the technological stack of the system. Their results show that the majority of the studies combine variations of MAPE, and based on a few studies from their survey, the authors identify three adaptation patterns with different combinations of multiple types of self-adaptation within the system. Their patterns distinguish between the type of adaptation mechanisms, their layer locations, and the cross-layer inter-adaptation interactions between the respective mechanisms. The three proposed patterns aim at tackling the following three problems: (1) a distributed application seeks to improve the utility of its services to the physical resources by dynamically exploiting rich context information, (2) a distributed application exploits data of individual resource to improve its overall utility by changing the resource configuration that produces the functionality of the application, and (3) a distributed application seeks to improve the overall utility of its service, which requires the autonomous entities to efficiently share information and coordinate their tasks on a local basis. In their study, the authors have explored how different adaptation mechanisms have been utilised for self-adaptation in CPS, without providing a common and shared understanding of what system adaptation means in the first place. Furthermore, the patterns identified in this work are not informative for the specific design of a SACPS, nor do they tackle concrete specificities and properties of the CPSs that are unique for these systems and differ, for example, from other distributed and decentralised applications or software systems.

Architectures. At the architectural level, Affonso et al. [14] and Braberman et al. [15] propose two reference architectures, which—to the best of our knowledge, are the only existing reference architectures in the literature that support more systematic guidance for the development of self-adaptive systems. Affonso et al. [14] present a reflection-based reference architecture for self-adaptive software, named RA4SaS, which aims at developing software entities that are transparently monitored and adapted at run-time. Namely, to perform these operations, this architecture proposes using modules in an "assembly line", which allows a software entity to be disassembled, adapted, and reassembled automatically by these modules.

Braberman et al. [15] present MORPH, a reference architecture that distinguishes the dependencies between structural reconfigurations and behavioural adaptations in self-adaptive systems. Concretely, the proposed architectural approach involves run-time change of the system configurations (e.g., the system's components and their bindings) and behaviour update (e.g., components orchestration, reactive behaviour, etc.) based on design time predefined configurations. In our logical architecture, compared to the reference architectures described above, the adaptation logic not only adapts the managed element(s) but also modifies and adapts itself during run-time.

Frameworks. The most prominent framework proposed in the literature for engineering self-adaptive systems is the Rainbow framework, developed by Garlan et al. [16]. Rainbow is a two-layered framework for architecture-based self-adaptation utilising utility theory. In the framework, the adaptation infrastructure is tailored using the system-specific adaptation knowledge, including the types and properties of components, behavioural constraints, and adaptation strategies. Rainbow uses specific adaptation mechanisms for planning (e.g., utility theory), which is something that we do not prescribe in our logical architecture and is left to the designers of the system. Furthermore, the adaptation logic of the framework is predefined during the design of the system, as Rainbow does not support run-time modification of the adaptation logic, which disables the approach to deal with uncertain situations and changing conditions that were not known during its design, as we do in this paper. A few other frameworks have also been proposed in the literature that apply to only a single technology, e.g., Java-based applications [17,18], or to a single domain, e.g., mobile applications [19,20].

Camara et al. [21] propose a framework for self-adaptation based on quantitative synthesis and verification that separates planning into architecture reconfiguration and task planning problems on a use case of mobile service robots—similar to ours. The authors have identified the problem of the large solution spaces while searching for the best combination of software architecture configuration and task planning specification for adaptation. This is because quantitative guarantees associated with reconfiguration and behaviour strategies depend on different information from various models that change at run-time. This potentially invalidates every quantitative guarantee associated with pre-computed strategies (the core limitation of the MORPH approach [15] presented above). This paper acknowledges the importance of information from models that change at run-time, and their proposed approach enables automated run-time decision-making for self-adaptation. In our architecture, similarly as in [21], we also promote both behavioural (the adaptation of the managed elements) and structural adaptation (the adaptation of the adaptation logic). However, our logical architecture resides on a higher abstraction level, and it is built while considering specific characteristics of CPSs that are entirely out of scope in the approach proposed in [21] (e.g., partiality and uncertainties of observations, dynamic context, need for

knowledge aggregation, etc.). Furthermore, our architecture is built within the proposed formalism for adaptivity, consequently supporting a more systematic development of SACPSs. Concretely, the paper from Camara et al. [21] mainly focuses on the planning phase of the self-adaptive system, and it could be seen as one potential technical solution for the planning component of our logical architecture (see Fig. 4).

To the best of our knowledge, our logical architecture is the only proposed architecture in the literature that can serve as a blueprint for engineering SACPSs. Concretely, engineering autonomous and decentralised SACPSs that make partial and uncertain observations of the dynamic, changing, and the uncertain context in which they operate. Since the context is changing unpredictably during run-time, a particular emphasis in the architecture is put on run-time knowledge derivation. Namely, the knowledge in the adaptation logic gets continuously updated, which should later enable a better adaptation of the overall SACPSs.

4. On context, adaptivity and self-adaptive systems

4.1. Defining context and the notion of partiality and uncertainty

Over the years, with the shifting focus on modern, dynamic systems, including CPSs, the importance of the notion of *operating context*, or simply *context* has emerged—in particular when understanding and engineering self-adaptive systems is in the focus [3,4,22]. According to traditional software engineering [23], every system operates in an environment, also referred to as universe ($\mathbb{U}$). The *context* ($\mathbb{C}$) is the part of the environment that consists of all the objects relevant to the system. The system-relevant parts of the environment affect the system input (I), as well as the state and the behaviour of the system.

$$I \subset \mathbb{C} \subset \mathbb{U}$$

Namely, every system is embedded in a context, which is everything that somehow interacts with the system and influences the system's input and behaviour. The system perceives, accesses, and interacts with its context through its inputs and outputs. Please note that *first*, not everything from the real context is necessarily perceived through the (encoded) system's inputs ($I \subset \mathbb{C}$), e.g. temperature, dust, humidity, or radiation but still has effects on the state of the system. And *second*, dynamic systems, e.g., CPSs that operate in changing, unpredictable, and unanticipated contexts cannot (fully) know or anticipate even the perceivable context during the system's design. As a result, regardless of the assumptions, the engineers make to model the context during the design of the system, the model will only be an approximation of the "real context", and will not fully reflect the actual context in which the system operates at run-time. This gives rise to the notion of *partiality* in the systems. The systems—primarily due to their input limitations—at a specific time, can only make *partial* observation of their context. *Uncertainty* is, therefore, an intrinsic property of CPSs. In sum, the reasons for the *partial* nature and *uncertainty* about the context are threefold: (1) the system may not be equipped with the necessary technology, i.e., lacks a concrete type of input (for example, no sensor capable of detecting surface conditions) (2) or the technology used may be limited in its capability, the input receives only a specific range of values (e.g., limited range of a sensor) (3) or the technology is imprecise and faulty.

In our running example robots detect elements in the room using a LiDAR sensor. Other conditions, e.g., wet floor surfaces that have relevant effects on the system, cannot be perceived by robots due to missing equipment, i.e., a missing humidity sensor. Furthermore, a LiDAR scanner has only a limited measuring range, for example, 10 m. Thus, at a specific point in time, there are tasks outside of the sensor's range that cannot be detected, although they affect the system's state and behaviour, and the fulfilment of the system's business and adaptation goals. Finally, due to sensor uncertainties, the robots might observe tasks on locations different from their real locations, failing to observe an existing task, or observe task in a location where there is none.

4.1.1. Context in the frame of the logical architecture

As previously said, as part of the logical architecture, we focus on optimising the traversal of CPSs with respect to adaptation goals, in two- or three-dimensional environments or contexts. This enables us to further narrow down the general definition of context. Namely, we define context $\mathbb{C}^d_P$ as a subset of $\mathbb{C}$ ($\mathbb{C}^d_P \subseteq \mathbb{C}$), *expressed as a function that maps 2D or 3D locations (depending on the considered dimensionality (d) of the context) to characteristics of the context at that location*. These characteristics originate from some set of possible properties $\mathcal{P}(P_{world})$ that may hold in the real world.

$$\mathbb{C}^d_P : \mathbb{R}^d \rightarrow \mathcal{P}(P_{world}),\ \mathbb{C}^d_P \subseteq \mathbb{C} \subset \mathbb{U}$$

The system can neither perceive nor anticipate all possible P_{world}. Instead, the CPS uses its sensors to produce partial and uncertain observations of the context, as previously discussed. These observations encode (approximations of) the real-world properties P_{world} as observable properties P_{obs}. Additionally, the model built in the adaptation logic infers properties of the world that cannot be observed directly and encodes them as model properties P_{mdl} (further explained in Section 7.2).

4.2. Defining adaptivity

In recent work, Kugele et al. [24] propose a taxonomy of autonomous systems, in which they formally specify different levels of autonomy. As part of this work, the authors reason that it is not trivial to classify a complete system as autonomous or non-autonomous. Instead, *autonomy is a property of individual functions*. Within the frame of self-adaptive systems, we adopt the same idea. Concretely, throughout the rest of the paper, the focus shifts from discussing the adaptivity of the system as a whole to the adaptivity of a specific system function sf. Furthermore, adaptation as a process always aims at improving or preserving certain adaptation goals (i.e., one or more quality objectives) in dynamic and uncertain internal and contextual conditions.

In the following, we define adaptivity formally: The behaviour of a system at a given moment in time is determined by its system function sf. For simplicity, we assume without loss of generality that the systems are deterministic. The function sf maps the input of the system $i \in I$ and its internal state $\sigma \in \Sigma$ and context $c \in \mathbb{C}$ to an output $o \in O$ and new state $\sigma' \in \Sigma$, defined as follows:

$$sf : I \times \Sigma \times \mathbb{C} \rightarrow O \times \Sigma,\ f(i, \sigma, c) = (o, \sigma') \tag{1}$$

Parts of the context are reflected in the system's encoded input, but there are also parts of the context that are implicit yet potentially impact the output. In turn, the context may be modified by the system's output, but since there are usually other influences as well, we do not need to consider it as part of the system functionality's output. We then define adaptivity as a sequence of functions $(sf_0, sf_1, \ldots, sf_t, \ldots)$ that evolve over time. These functions generally *improve* the business and adaptation goals at run-time by mapping a function sf_t to another "better" function sf_{t+1}. The notion of "better" is determined by a *Quality Function* Q, and it is used throughout the rest of the paper. However, defining the Quality Function Q, i.e., what precisely is meant by "better" and how it is measured, is use case-specific and it is left to the designers of the self-adaptive system.

The Quality Function is used to measure the quality of a system function sf_t at time t:

$$Q : (I \times \Sigma \times \mathbb{C} \rightarrow O \times \Sigma) \rightarrow [0, 1] \tag{2}$$

Q needs to measure the degree of fulfilment of the business and the adaptation goals at a specific point in time. Independent of the concrete realisation of the Quality Function Q, we stipulate its value to always reside within a fixed range normalised between 0 and 1. The lower boundary is defined by the point where the business goals are no longer met, whereas the upper limit is reached when all of the business and adaptation goals are achieved perfectly.

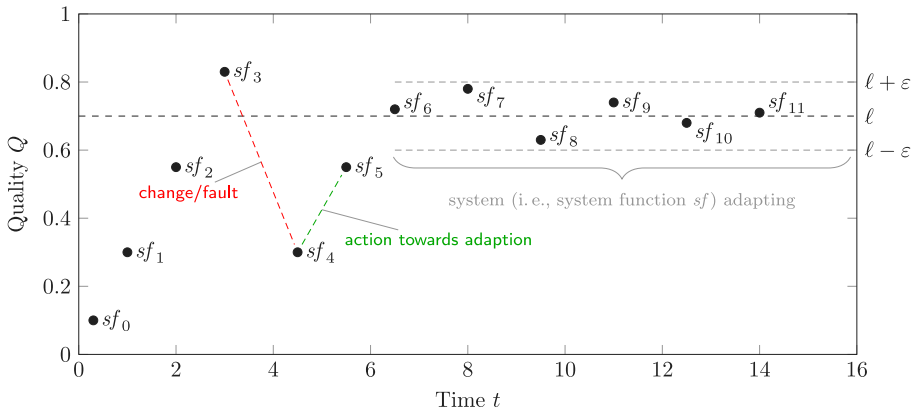


Fig. 3. Exemplary adaptation of the system (i.e., the system function sf) over time: $Q = 0$ indicates that the business goals of sf are not fulfilled and $Q = 1$ would represent a perfect fulfilment of the business and the adaptation goals. The system adapts close to a recorded maximum value ($Q(sf_3) = 0.83$), with $\ell = 0.7$ and $\varepsilon = 0.1$, after time $t = 6.5$.

Definition 1 (*Adaptive System*)**.** An *adaptive system* is an infinite sequence of functions $(sf_i)_{i=0}^{\infty} = (sf_0, sf_1, \dots, sf_n, \dots)$ that satisfies some quality objectives, s.t. the Quality Functions Q of the infinite sequence of functions converges in some sense, e.g. towards a limit ℓ s.t.

$$\exists \ell, \varepsilon, i \; \forall t \geq i : |Q(sf_t) - \ell| < \varepsilon, \quad \text{with } \ell, \varepsilon \in [0,1] \text{ and } 0 \leq \ell - \varepsilon < \ell < \ell + \varepsilon \leq 1 \tag{3}$$

where Q is a Quality Function of the system function sf_t at the specific point in time t. For this notion of convergence, after time i, the Quality Function of sf converges, and ℓ and ε are threshold values for the convergence: ℓ is the convergence limit and ε is the measure of the closeness of the convergence (see Fig. 3).

Note that the specific notion of convergence here is immaterial and that the simple definition used above does not allow for *infinite* repetitions of a pattern where a system operates normally (and improves its quality over time), then fails-operationally (and thus reduces its quality), then takes up normal operation again, fails again, etc. Hence, the Quality Function Q is not always a monotonically increasing function at each time step, especially when changes and uncertainties occur—either internal (from the managed element) or external (from the context)—including system's faults (see Fig. 3). Also the Quality Function Q of the system functions series converges to the limit l over time, but it does not necessarily converge to 1. In an optimal case, when a system adapts, its Q should converge close to a recorded maximum value (see $Q(sf_3)$ in Fig. 3). However, in case there is no other adaptation option available then the system can adapt to, e.g., a fail-operational state with a reduced Q and converges to a value smaller than the maximum existing Q.

Fig. 3 illustrates the convergence of the Quality Functions of a sequence of system functions sf_t close to a maximum value ($Q(sf_3) = 0.83$), which convergence limit $\ell = 0.7$ and measure of closeness $\varepsilon = 0.1$. Please note that in the figure only a part of the infinite sequence of system functions is shown. Also as indicated in the figure, there is an internal or external change, which causes the quality to drop at first. A subsequent action towards the adaptation, however, again achieves an increased quality performance. The Quality Function, the precise notion of convergence as well as the parameters have to be specified as part of the system's design process. We further specify how the Quality Function Q is calculated, considering different business and adaptation goals in Section 6.

Until now, in software engineering and in various other disciplines, the term adaptation has been used intuitively without deeper understanding and explanations of the precise meaning [25]. Nonetheless, if and when a system behaves adaptively cannot be answered by means of intuition, especially in engineering and science. Definition 1 and the introduction of Quality Function, contribute towards closing this gap. The Quality Function can be used with two purposes:

1. to *define* the system's adaptivity as explained in Definition 1 and exemplified in Fig. 3, and
2. to *measure* its (approximate) value at runtime. The measured, approximated value can be used to (i) to (passively) evaluate the system's adaptivity and (ii) to (actively) steer the adaptation and decide which adaptation actions shall be triggered based on the value of Q.

If the system itself does the latter, it can be considered an instance of goal-driven self-adaptation.

Note that the parameter of the Quality Function is *defined* as an entire function with potentially infinitely many inputs and/or contexts. This means that its value usually cannot be *measured* at runtime directly. However, its value can be approximated by considering information about the system's behaviour since the last adaptation step.

4.3. Characterisation of self-adaptive systems

The separation between system function (sf) and adaptation in the previous section sets the foundation for differentiating between (self-)adaptive systems, and the *ordinary* systems that are considered as non-adaptive. Therefore, while discussing self-adaptive systems it is essential to determine which aspects (which system function, adaptation goals (i.e., the quality objectives), and contextual and system situations (i.e., changes, uncertainties)) are relevant for the specific adaptation, based on which the adaptation logic for the concrete system (i.e., system function) is constructed. The existence of the adaptation logic enables the adaptation to be executed by the system itself, in an autonomous manner without any user interaction, and we refer to this case as self-adaptation. Conversely, if a human administrator triggers the adaptation, then we call this a manual adaptation, or system reconfiguration. Furthermore, in Section 3, we have discussed that according to [10] the four MAPE phases can be decentralised and distributed in the managed element (i.e., the system). In response, we conclude that the existence of knowledge in the adaptation logic, created w.r.t. the relevant aspects for the adaptation, is a *minimum requirement* for the system to be self-adaptive.

The concepts behind self-adaptivity can hardly be explained without considering the system's context and its interaction with the system, as previously explained in Section 4.1. Namely, (1) changes in the system itself (the managed element), (2) changes in the context, (3) or changes of the adaptation goals [3] may trigger the system to self-adapt. As a result, the knowledge in the adaptation logic stores the adaptation goals and the models of the context and the managed elements (i.e., the CPSs, see Fig. 1). Note that in this paper, we focus on self-adaptation triggered by system and context, where we assume the adaptation goals to be defined during the design of the system and to not change during operation of the system.

If the models of the context and the CPSs are hard-coded in the knowledge of the adaptation logic during the design of the self-adaptive system, then this is sufficient only if all their states can be fully anticipated during the system's design. However, due to uncertain, changing and dynamic conditions at run-time, the states of the context and the CPSs cannot be fully anticipated by the engineers of the systems during systems' design. Therefore, the models of the context and the managed elements cannot be hard-coded in the adaptation logic at design time, but instead, they need to be *models at run-time* (models@RT), which are continuously updated during the operation of the self-adaptive system [26,27]. If the underlying system or the context change, then their representations in the adaptation logic—the models—should also change. In this work, we focus on *context models@RT*, which are updated based on the information the self-adaptive system "learns" during its operation.

5. Overview of the logical architecture

This section introduces our logical architecture for engineering self-adaptive CPSs that operate in dynamic, uncertain, and partially observable context. In comparison with conceptual architectures or models (e.g., MAPE-K), logical architectures should provide a higher level of detail without restriction to any particular technology or implementation. A logical architecture [28,29] defines: (i) functional components in which different elements from a physical architecture are grouped by functionality (1-to-N or M-to-N mapping), (ii) arranged by their communications—how different components in the architecture interact and what information they exchange. Note that centralised components in a logical architecture could be distributed as part of a technical implementation.

Our proposed logical architecture, shown in Fig. 4, is motivated by and built upon the definition of adaptivity and the characterisation of self-adaptive systems from Sections 4.2 and 4.3, respectively. Additionally, it is specific enough to be useful in the design and the implementation of a class of use cases, as previously explained in Section 2. The main focus of our architecture is modelling and deriving knowledge about the context, which enables the adaptation logic to reflect the actual context during the systems' operation, which is dynamic and could not be anticipated and specified in the SACPS during its design. Concretely, our architectural solution can be seen as methodological guidance for dynamic knowledge management in the adaptation logic, i.e., it sets a foundation for procedures for reasoning and updating the context model@RT based on independent, partial and uncertain observations made by decentralised and autonomous CPSs. Note that each architecture component is mapped to one or more phases of the MAPE-K (marked with letters in the upper right corner of the components).

This section provides a high-level overview of (1) the design time components: the System Goals and the Quality Function (including the Quality Evaluator) in Section 5.1, and (2) the run-time components: the Managed Elements and the Adaptation Logic, in Sections 5.2 and 5.3, respectively. A more detailed explanation of the design time and the run-time components of the logical architecture are given in the following Sections 6 and 7.

5.1. System Goals, Quality Function and Quality Evaluator

The *System Goals* are based on domain knowledge supplied by the stakeholders or the designers of the self-adaptive system. They are defined during the design of the system, and in our work, we assume they do not change over time. In self-adaptive systems, we differentiate between two kinds of goals: *Business Goals* and *Adaptation Goals*. Business Goals are binary conditions based on the functional requirements of the system (i.e., the system function sf). Upholding them is the basic requirement for the performance of a self-adaptive system to be considered adequate. The Adaptation Goals are quantitative measures of the system performance based on its non-functional requirements, which are one or more quality objectives. Together the goals form the Quality Function Q (see Section 4.2) that the self-adaptive system strives to maximise. Thus, the Goals determine what behaviour the Planning Logic enforces and what information it requires to do so. This determines the kind of characteristics the Context Module has to track and model from the actual context. The Goals and the Quality Function are further defined in Section 6. Furthermore, as part of the self-adaptive system in our logical architecture, we propose the Quality Evaluator component, which is derived from the Quality Function Q. The existence of this component could potentially enable a goal-driven self-adaptation by actively guiding the required actions towards an adaptation, which will be decided based on the value of the Q, as previously described while discussing the purpose of the Q in Section 4.2.

In the running example the Business Goals are two-fold: robots keep a room clean, and robots do not collide. The Adaptation Goals relate to measures of *the quality of that cleaning process*. However, how this is defined and specified depends on the engineers of the self-adaptive system and their own interpretation of the *quality of that cleaning process*. For example, it can be calculated based on how many tasks in the room are cleaned while minimising the total distance travelled by the robots, or cleaning as many tasks in the shortest time possible, or minimising the time a dirt task remains on the floor before it is cleaned. In Section 6, we provide additional details on how the quality function Q is constructed considering the business and the adaptation goals, which we further concretise and evaluate for our running example in Section 8.1.

5.2. Managed Elements

The *Managed Elements* are any number of CPSs (one or more) that interact with the physical world, specifically their context. As such, they represent the only interface between the self-adaptive system and its context. They provide their observations of the context to the adaptation logic and, in turn, act on instructions received from the adaptation logic. The managed elements comprise the decentralised monitoring (M) and execution (E) steps of the MAPE-K loop. The managed elements are further explained in Section 7.1.

In the running example the managed elements are the robots deployed in the room. The robots detect the dirt tasks and perceive obstacles and the other robots, using LiDAR sensors in a decentralised manner. The observations from the monitoring are propagated to the adaptation logic. After analysing the observations from all the robots and consolidating them into knowledge, the planning component in the adaptation logic assigns the next (optimal) target locations to each robot. The navigation and the traversal to those locations is then executed in a decentralised manner by every robot independently. Note that the propagation, analysis, and consolidation of knowledge takes time. It is dependent on the number of robots. In case of abrupt events, this could be safety-relevant. However, each robot has knowledge of the local context and can therefore act without propagated knowledge and, for example, take evasive action or choose a different safety tactic to mitigate risk and reach a safe state (fail-safe strategy).

5.3. Adaptation Logic

The *Adaptation Logic* accumulates observations that each managed element independently makes of the context and uses this knowledge to come up with instructions for all the managed elements that optimise the achievement of the adaptation goal(s). The adaptation logic is itself subdivided into multiple sub-components: The *Context Module*, which is responsible for analysis and the acquisition of knowledge, and the *Planning Logic*, *which uses the knowledge to plan the actions and adapts the target locations that the managed elements should carry out*. Together these two components handle the processing of information and the adaptation at run-time, and they are explained in more detail in Sections 7.2 and 7.3.

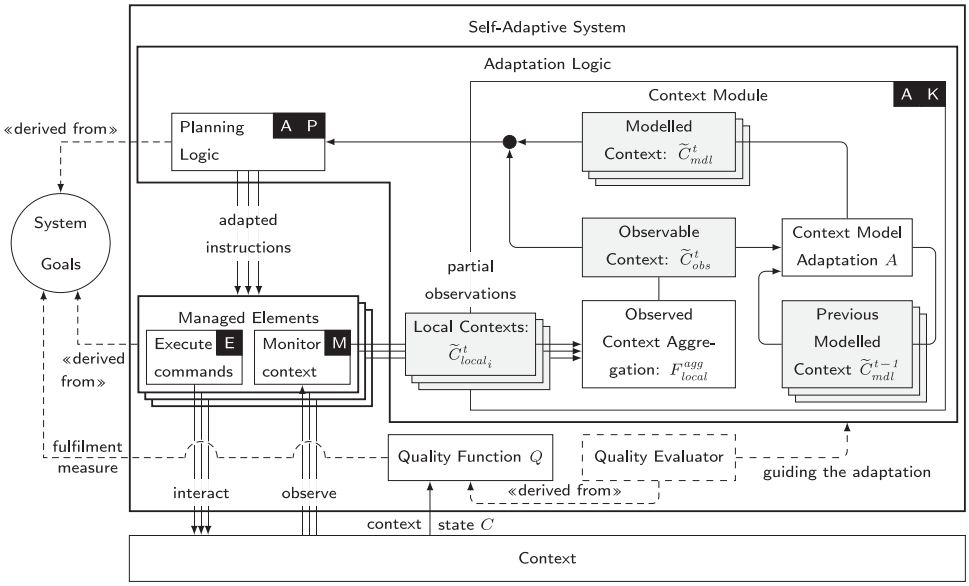


Fig. 4. The proposed logical architecture. The different components in the self-adaptive system are marked in order to signify where different phases of the MAPE-K loop are implemented. The grey boxes are not functional components; instead, they are the context models that can be seen as separate artefacts. We have kept them as separate components in the architecture, as they have a central role in the solution and clarify what information passes from one component to another.

5.3.1. Context Module

The *Context Module* is responsible for the analysis of the observations from the managed elements and subsequent deduction of knowledge about the context, which is later used as a base for decision making by the planning logic. At each time-step t, it performs these three tasks: It

1. collects local observations of the context made by each distributed Managed Element independently: the *Local Contexts* $\widetilde{C}^t_{local_i}$
2. combines observations from the different Managed Elements to a unified view of the context: the *Observable Context* $\widetilde{C}^t_{obs}$. This only covers the locations that are currently observed by the managed elements.
3. stores information for the observed and the unobserved part of the context: *Modelled Context* $\widetilde{C}^t_{mdl}$. Different $\widetilde{C}^t_{mdl_1}, \widetilde{C}^t_{mdl_2}, \dots, \widetilde{C}^t_{mdl_N}$ for encoding different aspects of the actual context might exist.

In the running example, the context is two-dimensional. We hence model it as a grid, in which each cell can be occupied by a dirt task, a wall, another robot, or be unoccupied. Therefore, the Context Module builds a grid map of the context and keeps track of which parts of the room are dirty by combining the observations of the robots. It uses the observations (current and past) by the robots to track the locations which are getting dirtier more often. Based on this history, it then approximates the underlying distribution and provides predictions for the dirtiness of unseen areas.

5.3.2. Planning Logic

The *Planning Logic* receives the most recent knowledge about the context. Given this information, it determines the actions for the managed elements that would further improve the goals. Concretely, the planning logic is designed such that it aims at continuously improving the Quality Function Q by adapting the target locations for all CPSs that collaboratively attain the task, based on the input from the Context Module. Its structure can be fixed at design time, in which case the self-adaptation of the system is exclusively performed within the Context Module during run-time. Moreover, the Planning Logic itself can change its algorithms, structure or parameters at run-time, based on the input it receives from the Context Module.

In the running example, this logic deploys the robots efficiently by providing them with appropriate target locations to navigate. It uses the robot locations and current observation of dirt, in combination with the learned past observations and predictions about the unseen areas to find the best adaptation action for each robot, i.e., to assign the robot to dirty area that will optimise the goals. As the learned models in the knowledge moves closer to the true distribution of the dirt, these decisions will lead to better results w.r.t. the goals.

6. Logical architecture: Design time components

In our logical architecture, the Goals and the Quality Function are design time components, which are use case-specific and guide the development of the other components in the self-adaptive system, i.e., the Managed Elements and the Adaptation Logic. Considering the Goals, the designer of the self-adaptive system needs to define the following: (1) the *Quality Function* Q (previously introduced in Section 4.2) as a measure of how good the system performance is with respect to the business and adaptation goals, (2) what behaviour of the Managed Elements the Quality Function Q improves, given the state of the context along with how the *Planning Logic* can determine what actions contribute to such behaviour, and (3) what information about the state of the context and the system is required by the Planning Logic, and how to extract this information from the observations made by each Managed Element.

6.1. Quality Function

For every business and adaptation goal, it needs to be defined what constitutes a good behaviour with respect to the specific goal. This enables us to reason what is “better” or how the Quality Function Q improves over time. We measure the quality of the behaviour of a self-adaptive system by evaluating the state of the context $\mathbb{C}$ and the system Σ during run-time. With C^t we label a “snapshot” of the context $\mathbb{C}$ at a particular point in time t ($C^t \in \mathbb{C}$). Note that this calculation of the Quality Function Q is an approximation, as described in Section 4.2, in which we have abstracted the input and outputs, since they are explicitly encoded in the system state.

Formally each goal g (business goal (g_b) or adaptation goal (g_a)) defines a *Goal Quality Function* q_g which maps the context state C (and

hence the input to the system) and the system state Σ at time t to a real-valued score, in the range between 0 and 1:

$$q_g^t : C^t \times \Sigma^t \rightarrow [0, 1].$$

The Quality Function Q for all the goals at a specific point in time (t) is calculated as follows:

$$Q^t = \prod_{g_b \in G_{business}} q_{g_b}^t(C^t, \Sigma^t) \cdot \left(\sum_{g_a \in G_{adapt}} \alpha_{g_a} q_{g_a}^t(C^t, \Sigma^t) \right) \quad (4)$$

where each q_g considers a pair of context and system states (C, Σ) at time t, and α_g is the weight associated with each adaptation goal. Since $Q = 0$ if one of the business goal (g_b) fails, all *Goal Quality Functions* q_{g_b} of the business goals are combined in a product and multiplied with the function results of the adaptation goals. The lack of fulfilment of the business goals indicates that the overall system is non-functional. As a result, any adaptation is not possible when the system fails to fulfil its basic functionality (i.e.. the system function sf) in the first place. In the following, we explain how the concrete *Goal Quality Functions* are calculated.

6.1.1. Business Goals

The Business Goals reflect the functional requirements of the system. If they are not fulfilled, it does not matter whether or not the behaviour of the self-adaptive system was optimal with respect to the adaptation goals. Therefore, for all Business Goals, the Goal Quality Function is defined as a binary distinction between the goal being fulfilled or not:

$$\text{for } g_b \in G_{business},\ q_{g_b} = \begin{cases} 1 & \text{if goal is fulfilled} \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

6.1.2. Adaptation Goals

Adaptation Goals reflect the non-functional requirements of the system and can be one or more quality objectives. The Goal Quality Function q_{g_a} for each Adaptation Goal is defined such that the score it defines increases when the system is doing "better" with respect to that respective goal. Furthermore, not necessarily all of the quality objectives are of the same importance for the system's stakeholders. Some quality objectives might be contradictory, e.g., minimising the wear on the mechanical parts can be opposed to getting things done as fast as possible, and some objectives cannot be achieved simultaneously. Therefore, each adaptation goal is also associated with a weight α_{g_a} such that $\forall g_a \in G_{adapt}$: $\alpha_{g_a} > 0$ and $\sum_{g_a \in G_{adapt}} \alpha_{g_a} = 1$. As a result:

$$0 \leq \sum_{g_a \in G_{adapt}} \alpha_{g_a} q_{g_a}(C, \Sigma) \leq 1. \quad (6)$$

Finally, after assigning corresponding weights α_{g_a}, and substituting Eqs. (5) and (6) in Eq. (4) it follows: $0 \leq Q \leq 1$.

The concrete realisation of the business and the adaptation goals for our running example, with their corresponding Goal Quality Functions and the calculation of the overall Quality Function Q, is given in Section 8.1, on the real implementation of the robotics system. Finally, please note that as part of this paper we only use Q to passively evaluate the system's adaptivity (as it will be further shown in the results section). Therefore, the Quality Evaluator is only a logical component in our architecture and we do not specify it further as part of this work, nor we specify how its feedback to the adaptation logic can guide the adaptation, which in theory would enable another dimension of goal-driven self-adaptation. As a result, this remains part of a future work (see Section 9.3).

7. Logical architecture: Run-time components

7.1. Managed Elements

The Managed Elements (i.e., the CPSs) are responsible for any direct interaction between the self-adaptive system and its context—the details of that interaction are encapsulated within their implementation. The Adaptation Logic makes no assumption about that implementation; therefore, it is independent of the actual type of the Managed Elements and only expects it to conform to the following interface:

Input from the Adaptation Logic. The adaptation logic sends to the Managed Elements adapted instructions, that request that the Managed Elements move to the new target location which is a result of the system's adaptation. After the adapted target locations have been assigned, the execution of the movement to those locations (e.g., motor actuation) is entirely up to the implementation of the Managed Element, which likely involves localisation and path planning, as well as sensing capabilities to avoid collisions with the other static and dynamic objects from its context. Please note that the Managed Elements do some "local" planning as part of the Execution component, which differs from the "global" planning in the Planning Logic of the adaptation logic. Namely, the "local" planning is in regard to planning the best trajectory to the received target location, and each Managed Element executes this independently. Whereas, the "global" planning adapts the whole system (all the CPSs) in response to the appearing tasks in the context, and assigns the adapted target location to all the Managed Elements according to the adaptation goals.

Referring back to Eq. (1), Eq. (2) and Definition 1, a system built according to our logical architecture is an adaptive system since we have a sequence of functions in which the state Σ of the system function sf is changed by the adaptation logic, whose components are further described in the rest of this section. Concretely, the adaptation logic builds knowledge over time, based on which the overall, internal state of the self-adaptive system changes, resulting in the same inputs (same observations from the context) to produce different outputs (adapted target locations) than what the managed elements would have produced in the absence of the adaptation logic, resulting in a sequence of different functions.

Output to the Adaptation Logic. *Local Context Observation* `observe_context()`: Returns a snapshot of the context at the current time step t from the point of view of a single Managed Element i—the *Local Context* $\widetilde{C}^t_{local_i}$. Each Managed Element uses whatever sensing capabilities it has to report on the part of the context it currently observes. These observations are the only source of information about the state of the context during run-time. Please note that these observations are uncertain and only partial. This output is refreshed at each time step t and used by the Context Module.

In the running example the robots operate in a two-dimensional environment. Therefore, whenever a robot receives a target location `(x,y)` from the Planning Logic, it uses Adaptive Monte Carlo Localisation (AMCL) [30] based on which the robot localises itself, and then accordingly plans and navigates a trajectory to its next location. While manoeuvring, the robot uses its sensors to avoid collision with dynamic obstacles like other robots or people moving through the room, and static obstacles, e.g., walls. At each time step, it reports information about its location, the partial space of the room it observes, and the detected tasks within this space, which are later aggregated in the adaptation logic.

7.2. Context Module

The Context Module produces the best possible representation of the context in which the SACPSs operate. To accomplish this, the Context Module builds knowledge which combines, aggregates and then stores the partial context observations made by each Managed Element. This

knowledge does not only contain the current state of the context ($\widetilde{C}^t_{obs}$) but also stores various aspects of interest from the context over time ($\widetilde{C}^t_{mdl}$), which later can be used as support for inferring and predicting properties of, for e.g., the unobserved parts of the context at a specific time t. The resulting representation spans on the entire 2D or 3D space of relevance, depending on the dimensionality of the operational context. The knowledge, i.e., the built context models@RT are later utilised by the Planning Logic.

Context Properties. The Context Properties encode every aspect of the context that is relevant for the decision making in the Planning Logic. They may be derived from the real properties of the context P_{world} but are independent of them. We differentiate between two types of Context Properties:

1. *Observable Properties* P_{obs} are directly observed by the Managed Elements. The observations represented by the Local Contexts $\widetilde{C}^t_{local_i}$ are expressed as Observable Properties and then passed to the Adaptation Logic. *In the running example*, observable properties are, among others, cell occupancy (i.e., the location of the dirt tasks).
2. *Model Properties* P_{mdl} represent assumptions about the context that are inferred from observations. *In the running example*, an important model property is the probability of a certain location (i.e., cell in the grid) to be dirty (i.e., occupied by a dirt task). There is a broad range of options what those probabilities represent and how they can be calculated. A simple example could be the probability of tasks appearance in that cell (calculated by dividing the tasks that have appeared in that cell with the total amount of tasks in the room).

For defining the logical architecture, we define the union of these two types of properties, as set of properties $P = P_{obs} \cup P_{mdl}$ needed to describe the relevant aspects of the context.

7.2.1. Local Context

At every time step, each Managed Element reports its own observation of the context as its individual *Local Context* $\widetilde{C}^t_{local_i}$. Each $\widetilde{C}^t_{local_i}$ assigns to each point in the line of sight of the respective Managed Element a set of observable properties P_{obs} and values for those properties. Concretely, it is a function that maps 2D (x, y) or 3D (x, y, z) coordinates to the properties at those coordinates. Please note that in the formulas, through the rest of the section, for simplicity we only use the 2D coordinates. For each location within the range of their observations, the Managed Elements record the values of the observable properties they can perceive. The output of the ith Managed Element at time t is therefore described as the following function:

$$\begin{aligned}\widetilde{C}^t_{local_i} : \mathbb{R}^d_{obs^t_i} &\to \mathcal{P}(P_{obs}),\\ (x, y) &\mapsto \{p \in P_{obs} \mid i \\ &\text{observed } p \text{ to be a property of point } (x, y)\}\end{aligned}$$

where $\mathbb{R}^d_{obs^t_i}$ is the part of the context observed by the ith Managed Element at time t and d is the dimensionality of the context. Individual observations from different Managed Elements can overlap, and due to sensor uncertainties they might even contradict with each other. Therefore, the individual observations need to be further processed and aggregated before they can be used to update the context model@RT.

7.2.2. Observable Context

The *Observable Context* $\widetilde{C}^t_{obs}$ is the unified and aggregated view of the individual observations from all the Managed Elements. It represents the model@RT of the most complete state of the actual, current context in which the Managed Elements operate. In other words, the most complete, aggregated observation of the context that all the Managed Elements make together at a specific time t. $\widetilde{C}^t_{obs}$ contains all observable information about the context, represented as observable properties P_{obs}, at the current time step t.

$$\begin{aligned}\widetilde{C}^t_{obs} : \bigcup_{i=1}^{n} \mathbb{R}^d_{obs^t_i} &\to \mathcal{P}(P_{obs}),\\ (x, y) &\mapsto \left\{p \in P_{obs} \mid p \text{ is a property of point}(x, y)\right\},\end{aligned}$$

where n is the number of CPSs (e.g., robots).

To get such a unified view, the individual Local Contexts have to be combined or aggregated first. The Observable Context Aggregation function F^{agg}_{obs} maps the n independent Local Contexts $\widetilde{C}^t_{local_i}$ of the Managed Elements to the unified $\widetilde{C}^t_{obs}$:

$$\begin{aligned}F^{agg}_{obs} &: \left(\mathbb{R}^d_{obs^t_i} \to \mathcal{P}(P_{obs})\right)^n \to (\bigcup_{i=1}^{n} \mathbb{R}^d_{obs^t_i} \to \mathcal{P}(P_{obs}))\\ \widetilde{C}^t_{obs} &:= F^{agg}_{obs}\left(\widetilde{C}^t_{local_1}, \widetilde{C}^t_{local_2}, \ldots, \widetilde{C}^t_{local_N}\right)\end{aligned}$$

This function has to account for all discrepancies which possibly exist between the Local Contexts. Concretely, the Managed Elements have sensors that are exposed to different run-time uncertainties, e.g., sensor failure, noise and imprecision [5]. Therefore, different Local Contexts might assign different values to the same property at the same location, for example, when one Managed Elements detects a task at a specific location, and another Managed Element does not. Concretely, if there is a semantics inherent to the properties, it might be violated by the combination of the Local Contexts assigning contradictory properties. How the aggregation function F^{agg}_{local} solves these conflicts is a technical challenge on its own, and it is up to the designers and the engineers of the self-adaptive CPSs, and depends on the needs of the use case. As a simple solution, one of the CPSs could be assigned as "lead", and in case some conflicts happen the lead's opinion can overwrite the rest. Also some peer-to-peer negotiation algorithms can be implemented. Furthermore, in [31,32], the authors propose a Subjective Logic-based [33] approach for aggregating context observations.

7.2.3. Modelled Context

The *Modelled Context* $\widetilde{C}^t_{mdl}$ accumulates observations over time, and it is the part of the Context Module that serves as a basis for "learning" the context. As part of the Modelled Context, the past (aggregated) observations are stored as knowledge. As previously explained in Section 1, different models can depict different aspects of the actual context. As a result, different Modelled Contexts can exist $(\widetilde{C}^t_{mdl_1}, \widetilde{C}^t_{mdl_2}, \ldots, \widetilde{C}^t_{mdl_N})$, and depending on what kind of knowledge is encoded in them, they can allow the system to make predictions about the state of unobserved parts of the context. The *Modelled Context* assigns both observable and model properties based on the knowledge the system has gathered over time. The result is a function defined on the entire context $\mathbb{C}^d_P$ that assigns every point in the context to a set of properties.

$$\begin{aligned}\widetilde{C}^t_{mdl} : \quad \mathbb{C}^d_P &\to \mathcal{P}(P),\\ (x, y) &\mapsto \{p \in P \mid p \text{ is predicted to be a property of point } (x, y)\}\end{aligned}$$

Each Modelled Context is updated based on the current Observable Context $\widetilde{C}^t_{obs}$ and the corresponding Modelled Context in the previous time step $\widetilde{C}^{t-1}_{mdl}$. This is done as part of the Context Model Adaption A, which accumulates the observations as knowledge and stores them into the Modelled Contexts $\widetilde{C}^t_{mdl}$. Concretely, A tracks the changes in the Observable Context $\widetilde{C}^t_{obs}$ and uses those to infer information and, consequently, to refine the Modelled Contexts.

$$\begin{aligned}A &: (\mathbb{R}^d_{obs^t} \to \mathcal{P}(P_{obs})) \times (\mathbb{C}^d_P \to \mathcal{P}(P)) \to (\mathbb{C}^d_P \to \mathcal{P}(P))\\ \widetilde{C}^t_{mdl} &:= A(\widetilde{C}^t_{mdl}, \widetilde{C}^{t-1}_{mdl}),\end{aligned}$$

where $\widetilde{C}^1_{mdl}$ is the initial design time model that the engineer of the self-adaptive system encodes in the adaptation logic. All of the above-mentioned allows the adaptation logic to take into account parts of the

context that the Managed Elements do not currently observe. Finally, all the context models@RT $\widetilde{C}^t_{obs}, \widetilde{C}^t_{mdl_1}, \widetilde{C}^t_{mdl_2}, \ldots, \widetilde{C}^t_{mdl_N}$ are then passed to the Planning Logic. Models of sufficient quality allow the Planning Logic to take into account how the context develops in the near future.

In the running example as previously explained the room is divided into discrete grid cells. Therefore, one value for each property is assigned per cell and maps all the points `(x, y)` within each cell to the respective properties of that cell. The Observable properties P_{obs} are:

- `isObstacle:bool`: cell is occupied by an obstacle e.g. a wall. This is mutually exclusive with the other two;
- `isDirty:bool`: the cell was discovered to be dirty, i.e., occupied by a task;
- `occupiedByRobot:int`: the cell is currently occupied by a robot and which robot it is.

To combine the $\widetilde{C}^t_{local_i}$ from different Managed Elements, a simple rule for reasoning, as part of F^{agg}_{obs}, is used:

1. `occupiedByRobot` is reported by each robot for itself and trusted as true,
2. if any $\widetilde{C}^t_{local_i}$ has `isObstacle` or `isDirty` for a cell then this is propagated to C_{obs},
3. the latest property issued per cell is kept as final (a naive way for conflict solving).

As a result, $\widetilde{C}^t_{obs}$ contains the detected dirt tasks, the true locations of the robots and all the information about the surrounding obstacles that are currently observed or seen at a time t. Based on $\widetilde{C}^t_{obs}$ different Modelled Contexts $\widetilde{C}^t_{mdl_1}, \widetilde{C}^t_{mdl_2}, \ldots, \widetilde{C}^t_{mdl_N}$, can be constructed—depending on what the models represent, or what information do they store, for example, accumulated task appearances, patterns of task appearances, probability distributions of the tasks, probability of a task appearing in a cell in the next time step, etc. This information is used to infer whether locations that have not been observed for a while are dirty or not. Consequently, the Modelled Context $\widetilde{C}^t_{obs}$ learns the underlying distribution of dirt appearance, for both, the seen and unseen areas at time t, and it is adapted based on the Observable Context $\widetilde{C}^t_{obs}$ and the Modelled Context from the previous time step $\widetilde{C}^{t-1}_{mdl}$. An example for a set of properties P_{mdl} necessary to model the above-mentioned distribution could be the following:

- `dirtPrior:float`: probability of dirt appearing in this cell at any given time.
- `dirtProb:float`: probability that a unobserved cell is currently dirty.
- `lastSeen:int`: number of time steps since this cell was last observed by a Managed Element.

The values for `dirtPrior` are deducted from tracking the appearance of dirt over time. By combining them with the information from `lastSeen`, the probability of an unseen cell being dirty at the current time step can be estimated.

7.3. Planning Logic

At each time step t, the Planning Logic takes into account the current state of the Observable Context $\widetilde{C}^t_{obs}$ and the Modelled Contexts $(\widetilde{C}^t_{mdl_1}, \widetilde{C}^t_{mdl_2}, \ldots, \widetilde{C}^t_{mdl_N})$ and determines the commands for all the Managed Elements:

$$PL : (\widetilde{C}^t_{obs}, \widetilde{C}^t_{mdl_1}, \widetilde{C}^t_{mdl_2}, \ldots, \widetilde{C}^t_{mdl_N}, \mathscr{S}^{t-1}_{\text{PL}}) \mapsto \langle c_1, \ldots, c_n \rangle$$

where $c_i \in (\mathbb{R}^d \cup \{\perp\})$ are the coordinates of the adapted commands for Managed Element i. An entry of $\perp$ signifies that no new command is sent to that Managed Element at this time. $\mathscr{S}^{t-1}_{\text{PL}}$ is the previous state of the Planning Logic (e.g. task queues or currently executing movement commands). The Planning Logic is characterised as that it maximises the quality function Q when given optimal $\widetilde{C}^t_{obs}$ and $\widetilde{C}^t_{mdl}$ that represent the most complete knowledge of the real context. At this high level of abstraction the Planning Logic is solely defined by this characterisation because everything else is highly dependent on the use-case and the goals.

8. Evaluation

To evaluate the applicability and the usability of the logical architecture, we have designed and implemented: (i) a *ROS-based, simulated, multi-agent SACPS* motivated by the running example from Section 2.3, and (ii) a so-called *evaluation framework* that can evaluate the adaptivity of the simulated SACPS within our defined use case.[2] The ROS-based, multi-robot system is simulated in Gazebo [34]. Referring to the logical architecture in Fig. 4, the simulated ROS-based SACPSs constitutes the self-adaptive system component, whereas the evaluation framework comprises the Goals, the Quality Function Q, and the context. Additionally, the evaluation framework can be used for any self-adaptive system that adopts the standardised interface of the framework, and the calculation of the Quality Function Q in our evaluation framework is universal (i.e., independent of the use case).

In this section, we first validate the integrated Goals and Quality Function Q in Section 8.1, before evaluating the logical architecture in Section 8.2.

8.1. Evaluation of the Goals and the Quality Function

While explaining the business and adaptation goals prior in Section 5.1, we already gave an intuition and briefly exemplified how they can be calculated for our specific running example. How specific the business and adaptation goals are defined and specified depends on the engineers of the self-adaptive systems. In this section, we first explain a concrete realisation of the business and the adaptation goals based on our interpretation of the *quality of the cleaning process*, using the formalisms from Section 6. This enables us to validate the concepts of Goals and the Quality Function, which are at the heart of defining adaptivity and a major part of our logical architecture for engineering SACPSs, on the actual implementations of the SACPS and the evaluation framework.

8.1.1. Goals and requirements for the SACPS in natural language

Initially, goals are expressed in a natural language and describe the intention of the engineers of the SACPS. The goal of our use case is minimal: the robots to clean the room. Depending on the system, this overall goal can be refined and extended with more detailed requirements for both the business and the adaptation goals.

In our running example, we define the business goals g_b, with their corresponding business Goal Quality Functions q_{g_b} as follows:

g_{b_1} The SACPS should work as expected regarding the navigation and the movement. No robot should have a collision; $q_{g_{b_1}}$ `[no_crashes]`

g_{b_2} The SACPS should work as expected regarding the completion of the tasks. This means that a minimum number of dirt tasks needs to be cleaned to consider the system as functional; $q_{g_{b_2}}$ `[minimum_cleaned]`

In addition to keeping the room clean, we also want to improve the quality of the cleaning process (the adaptation goal), despite the different run-time system and context uncertainties. In response, we define the following adaptation goals g_a, with their corresponding adaptation Goal Quality Functions q_{g_a} as follows:

[2] The source code of the complete implementation of the multi-robot, ROS-based SACPS and the evaluation framework, as well as the installation instructions, are available on the following links: https://github.com/tum-i4/SACPS-robotics-system, https://github.com/tum-i4/SACPS-evaluation-framework.

g_{a_3} The SACPS should clean as much dirt as possible; $q_{g_{a_1}}$ [higher_cleaning_rate]

g_{a_4} The SACPS should detect as much dirt as possible; $q_{g_{a_2}}$ [higher_detection]

g_{a_5} The SACPS should not ignore dirt; $q_{g_{a_3}}$ [shorter_dirt_existence]

g_{a_6} The SACPS should minimise the travelled distance; $q_{g_{a_4}}$ [less_distance]

How different goals are translated into Goal Quality Functions q_g and is explained in the next section.

8.1.2. Creation of Goal Quality Functions q_g

In the previous section, we described the goals of the SACPS in natural language, similar to something that a stakeholder would provide. To evaluate a SACPS according to these goals, we need to translate them into Goal Quality Functions q_g based on which the overall quality function Q is calculated. In the following, we give an example for the specification of the Goal Quality Functions for the second business goal (g_{b_2}).

The Goal Quality Function $q_{g_{b_2}}$ for g_{b_2} is specified and realised as follows: Since the second business goal g_{b_2} requires a minimum number of dirt tasks to be completed in order to consider the SACPS as functional, first we needed to interpret the meaning of this goal for our system and based on that interpretation, specify the Goal Quality Function.

Description: *"10% of the total dirt tasks must always be cleaned (at least) after 5 tasks have been initially spawned in the room"*.

We specify this function based on the total number of spawned tasks (`spawned_dirt_number`), the number of cleaned tasks (`finished_dirt_number`), and as a parameter for the minimum percentage of cleaned tasks we chose 10%. `minimum_cleaned`—which is another name for the Goal Quality Function $q_{g_{b_2}}$—then calculates the percentage of the number of cleaned to spawned dirt tasks, and returns 1 if the calculated percentage is higher than or equal to the specified parameter (the system fulfil its business goal), otherwise 0 (the system does not fulfil its business goal).

$$q_{g_{b2}} = \begin{cases} 0 & \text{if } (\texttt{spawned\_dirt\_number} \geq 5 \\ & \text{and } \texttt{finished\_dirt\_number} < 0.1 \cdot \texttt{spawned\_dirt\_number}) \\ 1 & \text{otherwise.} \end{cases}$$

If there are more business goals, the process would always be the same: interpret what input metrics are needed for a specific Goal Quality Function and then combine them to get a test value for a binary check that results in either 1, if fulfilled, or 0, otherwise.

Finally, the Goal Quality Functions for the Adaptation Goals need to be weighed out according to Eq. (6) in Section 6, and all the Goal Quality Functions are combined according to Eq. (4) in Section 6, which results in the overall Quality Function Q, based on which the adaptivity of the system is evaluated. Due to the space limitation, the specifications of the Goal Quality Functions for the rest of the business (`no_crashes`) and adaptation (`higher_cleaning_rate`, `higher_detection`, `shorter_dirt_existence`, `less_distance`) goals are out of the scope of this paper. However, all of them can be found in the source code of the evaluation framework and are included in the evaluation in the following section.

8.1.3. Validation of the Quality Functions Q

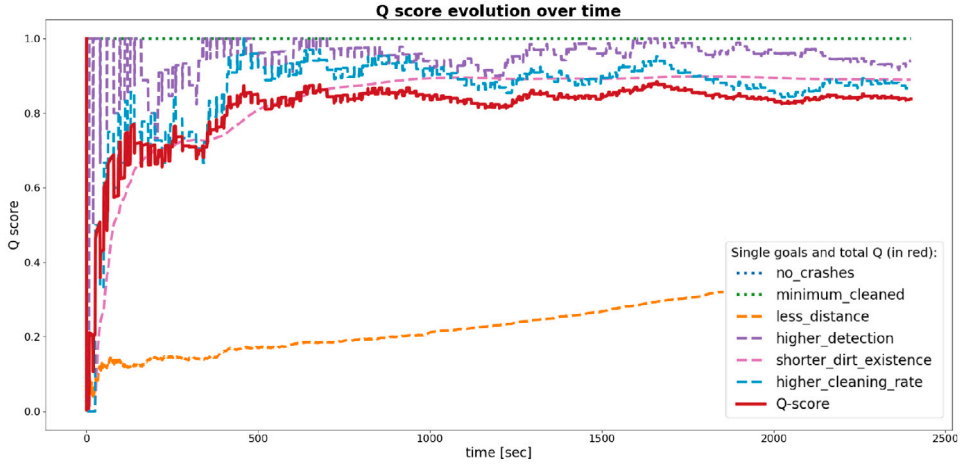
In this section, we validate the calculation of the Quality Functions Q, if it works as intended and if it enables us to evaluate the adaptation of the SACPS according to the specified goals, which were explained in the previous section for the concrete use case. Furthermore, through these results, we support or falsify the hypotheses from Section 1. To achieve this, we have conducted an extensive evaluation; however, we will analyse the calculated Q scores from only a few experiments due to space limitations. For all the experiments we have kept the same experimental setup: same room map (including room resolution and initial robot positions), two robots, a uniform distribution of the dirt tasks, and a simulation time of 2400 s. Before we started the analysis of the experiments, we also conducted repetition tests to show that the Q scores are reproducible and not random. Therefore, we ran the experiments five times with the same parameters. The Q score across multiple runs varied within a very small range of ± 0.03, which was expected due to the non-determinism of the Gazebo simulator.

To validate the system's adaptation and the specified adaptation goals, and to show that in order to argue if a system is adaptive or not, it needs to be considered inseparably from its context and the changing context situations, we have set up two different tests, in which we varied the spawning rate of the tasks for the robots. In the first test scenario, we have set up a lower spawning rate (see Fig. 5a), or in other words, the contextual situation in which the system can behave adaptively (Q converges close to the maximum value over time), according to Definition 1. In principle, we also have a sequence of functions, since at each timestamp in the graph, the system function is calculated based on internal system states that change over time (depending on the knowledge from the adaptation logic). Depending on the different models that are stored and continuously updated in the knowledge of in the adaptation logic, the robots get different, adapted target locations (outputs), despite making same observations from the context (inputs). Additionally, in contrast to the first test scenario, in the second test scenario, we have set up a high(er) spawning rate, and according to these context situations, the system does not adapt since there is a continuous drop in the Q (see Fig. 5b). Concretely, in the second test scenario, the spawning rate of the tasks in the room is so high that the robots—no matter how well they self-adapt—are still not be able to keep up with the adaptation goals (i.e., the quality of the cleaning process) due to the limited resources (e.g., the number of robots in the room or their velocity).

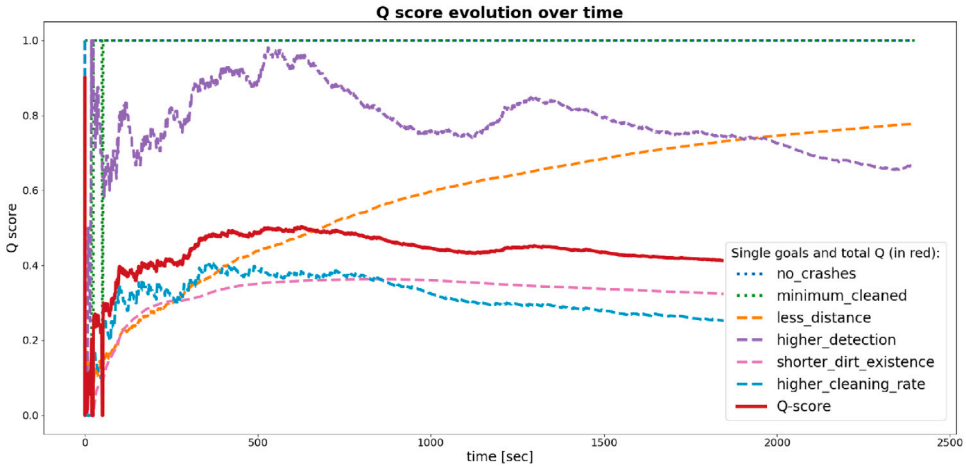
Furthermore, as previously explained, the Q score is always between 0 and 1, where 0 means the business goals are not met, and 1 means the business and the adaptation goals are achieved perfectly (see Section 4.2). As indicated, the analysis is not about the absolute Q scores at a specific point in time; but rather about the trend of the Q over time. While arguing if a system is adaptive or not, it must always be accompanied by the time period in which the system is adaptive. Concretely, if we only observe Fig. 5b for the first 500 s, we might label the system as adaptive, which is incorrect if we observe the Quality Function Q for a longer time.

To validate the business goals, in one of the test scenarios in our experiments, we increased the spawning rate so high that the robots could not keep up with the minimum threshold of 10% that we have specified in the `minimum_cleaned` Goal Quality Function. The evolution of the overall Quality Function Q, as well as the Goal Quality Functions for all the goals (business and adaptation), can be seen in Fig. 6a. Similarly, for the first Business Goal, in one of our testing scenarios, a robot crashes. As a result, it can be observed in Fig. 6b the Q score drops to 0, although the scores of the other Goal Quality Functions remains intact.

Main findings: Based on (1) our formal definition on adaptivity from Section 4, (2) the identification of the system function (sf) and how functioning differs from adapting, (3) and the conducted experiments, we were able to prove the hypothesis from Section 1. Namely, to argue if a system is adaptive or not, it needs to be put in a proper framing:
First, the adaptation goals (i.e., one or more quality objectives) for the specific system (i.e., system function) need to be identified and specified. Based on these goals, the adaptation logic for the concrete system function (sf) is constructed.



(a) Q scores with low spawn rate/the system is adaptive for the concrete context situation.



(b) Q scores with high spawn rate/the system is adaptive for the concrete context situation.

Fig. 5. *Q* score comparison regarding different contextual situations, i.e., dirt spawn rates (every other parameter is identical). The overall *Q* score is solid red, its Business Goal Quality Functions are dotted and its Adaptation Goal Quality Functions are dashed.

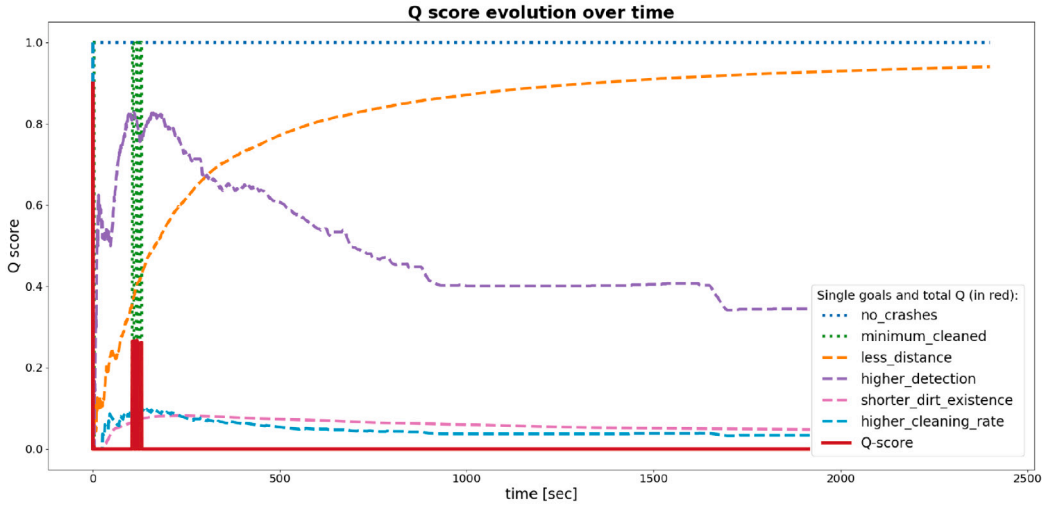
Please note that the same system (i.e., the system function) might be adaptive according to one adaptation goal and maladaptive according to another.

Second, the system (i.e., the system function) can only adapt to certain context (or system) situations (e.g., different changes and run-time uncertainties). No system universally behaves adaptively in every condition, at least in regard to a particular goal. To some contextual situations, the system might be adaptive, and to others, it might not.

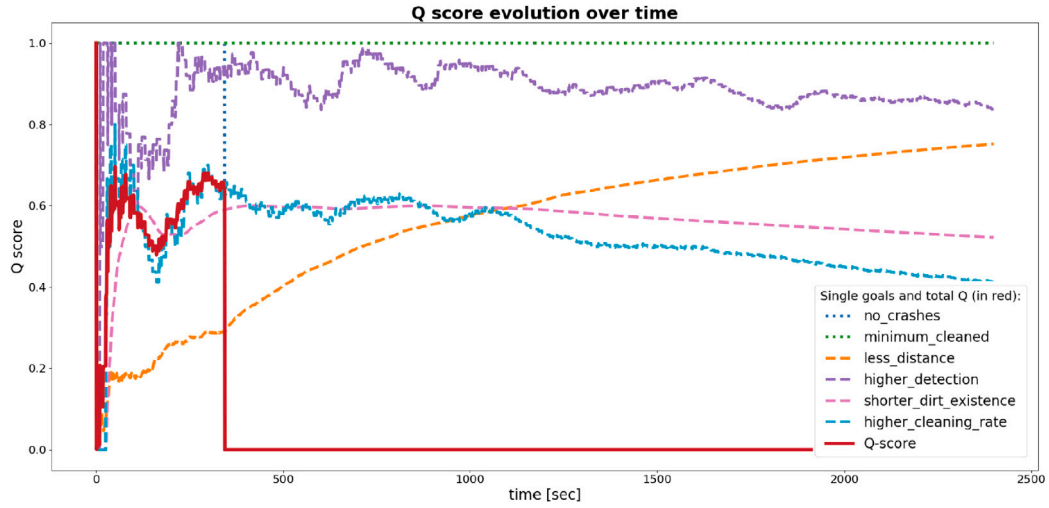
Finally, considering the time frame in which a system is adaptive is crucial. The absolute value of the Quality Function *Q* at one timestamp gives no information regarding the system adaptation. Therefore, we cannot observe a system at one point in time and call it adaptive. Adaptivity is bound to a specific time frame, i.e., the only way to evaluate if a system adapts or not is to observe over a certain period and observe how the quality objective(s) develop over the same time period.

8.2. Benefits of our Adaptivity Definition and the proposed Logical Architecture in comparison to the MAPE-K conceptual model and existing words

The conceptual MAPE-K model gives some intuition behind the engineering of self-adaptive systems, mainly by the separation of concerns between the managed element and the adaptation logic. However, the conceptual model is very general, and a more specific semantics of its two components and their interaction is still missing and is left on the engineers of the self-adaptive systems, potentially opening space for various misinterpretations. This is, on one side beneficial, since the same model can be used as a backbone for designing various types of systems. However, on the other side, its generality does not allow the conceptual model to be specialised according to some properties and characteristics that are unique for a specific type of systems. For example, the dynamic and changing context, and run-time sensor uncertainties and partiality in the observations in the case of CPSs. Furthermore, although the different MAPE-based design patterns proposed in the literature (e.g., [10]) might be more informative regarding the system's design; however, they (1) completely exclude the knowledge component that we have identified as a minimal requirement for a system to be self-adaptive (see Section 4.3), and (2) have the same limitations as the MAPE closed feedback loop itself. For example, their high level of abstraction, without providing any characterisation of how



(a) Q score for too high spawning rate (achieve 10% cleaning rate is not possible).



(b) Q score when one of the robots crashes at around 300s.

Fig. 6. Scores when a business goal cannot be fulfilled and overall Q score drops to 0. The overall Q score is solid red, its Business Goal Quality Functions are dotted and its Adaptation Goal Quality Functions are dashed.

a system built upon the MAPE loop differentiates from the *"ordinary"*, non-adaptive systems. Please note that our logical architecture itself is built upon the Master–Slave pattern proposed by Weyns [10].

In the following, we briefly summarise the benefits of our proposed logical architecture in comparison to existing works:

(1) our logical architecture is based on our formal definition of the system's adaptivity. The formal adaptivity definition enables us not only to measure if a system is adaptive and in which conditions but also to compare different self-adaptive systems,
(2) the differentiation between the system function (sf) and system adaptation enables us to differentiate between the ordinary systems and self-adaptive systems,
(3) identifying the minimal requirement for a system to be self-adaptive and the identified necessary framing (from the main findings) give further insights on the semantics of the adaptation logic and how it should be constructed,
(4) our logical architecture provides support for engineering systems that operate in dynamic and changing context, from which the CPSs make partial and uncertain observations. Concretely, our logical architecture enables the SACPS to change its adaptation logic and to reflect the actual context during run-time, which is something that none of the previous reference architectures [14,15] for engineering self-adaptive systems has proposed before, and
(5) finally, as previously explained in Section 5, the logical architecture defines how different components interact and what information they exchange. This provides another dimension of semantic precision, which has not been considered in the previous works [6,10,11,13,16].

As previously elaborated and discussed in Section 3 and various previous sections, to the best of our knowledge, there is no other (logical) architecture proposed in the literature that addresses some of the limitations of the MAPE-K conceptual model and MAPE-based patterns, and is specialised for a class of systems from the domain of CPSs. We hope that the architecture proposed in this paper will serve not only as a blueprint for engineering SACPSs but also as a baseline for future research. Concretely, as a baseline for comparison and evaluation of contributions and architectures proposed in the future.

9. Discussion

9.1. Threats to validity to the Quality Function Q

First, for calculating the Quality Function Q, we assumed that the context $\mathbb{C}$ and the system Σ are available. However, due to the partiality in the observations of the CPSs, the systems do not have access to the complete state of the context. Therefore, to quantify the quality of the self-adaptive system, there are two possible alternatives: (1) some form of external validation—which is not a subject to the same limitations of partiality—that provides the actual total state of the context. In practice, having this external observation is easily possible in an experimental setup or a simulation; (2) or incorporation of a business goal that includes metrics for context coverage percentage, that is later appropriately integrated in different goal quality functions q_g, in setups where the external validation is not available. We assume that we simulate the robotic system in all the given examples from our running example; therefore, the simulation inherently provides the external context validation. Second, the weighted sum method in the adaptation goals unfolds the problem of coming up with "good" α_{g_a} weights, which itself leads to a need for a multi-criteria optimisation problem, e.g., Pareto front optimisation.

9.2. Discussion on system implementation guided by our logical architecture

There are some aspects which might technically present a problem or a challenge on its own, but do not represent an issue on the abstraction of a logical architecture and therefore have not been considered earlier. Nevertheless, we want to briefly summarise some points of relevance, important while instantiating our architecture to a technical implementation.

So far, we only have one technical implementation of the robotics system, guided by the proposed logical architecture. As previously explained in Section 8, our technical implementation of the multi-robot system is ROS-based and simulated in Gazebo. Our implementation enables us to simulate any number of robots in any context (i.e., any room) with different static (e.g., walls, corridors, furniture, etc.) and dynamic objects (e.g., humans, or other robots with different goals), including the tasks that continuously appear for the robots in the room. The robots that we simulate are *TurtleBot 3 Burger*.[3] Furthermore, Gazebo relies on well-established physics engines, which enable simulations with high fidelity that closely resemble real-world robotics systems and their context, with a physically correct representation of the robots, including their size and volume, frictions, as well as their sensors and actuators [32]. Although we aimed to build as realistic simulation as possible, with a codebase that can be easily deployed on real robots, one aspect that our implementation does not consider is simulating the networking or the communication capabilities between the robots (i.e., the information exchange between different robots happens instantaneously). This aspect does not have any relevance as part of our logical architecture, but indeed it might have a range of different technical implications.

In sum, while discussing different technical implementations for other use cases that are instantiated from our logical architecture, some implementation-related questions will arise and need to be conveniently discussed and considered by the developers of the self-adaptive systems. For instance, (1) the communication of the CPSs (including the communication latency), (2) the type and the complexity of the task planning algorithms (e.g., for some optimisation-based algorithms, the CPSs themselves might have limited computational capabilities, in which real-time optimisation is not feasible at all), (3) the scalability of the implementation of different components, and (4) the models, concretely the models@RT, their complexity, and how they are stored, versioned, managed and updated during run-time.

[3] https://www.turtlebot.com/.

9.3. Discussion on future research directions

Although the procedure established in our logical architecture can be seen as methodological guidance for knowledge derivation and management in self-adaptive CPSs, it is essential to emphasise that we neither prescribe how to represent the knowledge in the adaptation logic, nor how to design and select the models the knowledge contains. Also, we did not elaborate further on the run-time uncertainties, the knowledge aggregation, nor the planning based on the different models@RT. Therefore, the logical architecture only establishes a blueprint of what processes are necessary and not how to accomplish them. As part of our running example, we proposed simplistic solutions to some of the open problems, which were necessary for depicting and exemplifying the contribution of the logical architecture itself; however, many problems still remain open:

1. How to represent the knowledge in the adaptation logic?

Knowledge representation is an essential concern for building self-adaptive systems, also acknowledged by Weyns et al. [10]. However, there is a scarcity of approaches for modelling knowledge in the adaptation logic, with a particular emphasis on modelling the context in the knowledge. Instead, in the existing works, knowledge modelling has been typically treated as a domain-specific task, later leading to ad-hoc solutions to observations aggregation.

2. How to aggregate partial and uncertain observations made by the decentralised CPSs? What are the uncertainties for the CPSs? How to represent uncertainties? How to reason about the uncertainties?

To update the models@RT—and accordingly, the knowledge component in the adaptation logic—the need for run-time information aggregation and reasoning emerges of what each CPSs individually observes. The CPSs are exposed to various run-time uncertainties: internal, for example, sensor failure and ambiguity, and external, for example, changes in their operational context. Consequently, there is a need for capturing uncertainty that will allow reasoning by an effective aggregation of the partial and uncertain observations made by the decentralised CPSs, based on which the models@RT are updated and later used in the planning phase of the MAPE-K.

3. How to plan and find the most optimal actions for the self-adaptive CPSs considering different models@RT?

To the best of our knowledge, utilising and combining different models@RT for analysis and planning of the next best adaptation actions, for the SACPSs is still under-researched.

4. How can the Quality Evaluator component guide a goal-driven self-adaptation?

As part of this paper, we used the value of the Quality Function Q to passively evaluate the system's adaptivity and we did not elaborate further nor specified how the Quality Evaluator component can actively guide and steer the self-adaptation of a system based on the value of Q. At its nature, this is an optimisation problem, and opens new directions of research in this field.

10. Conclusion

The lack of commonly accepted definitions of different terminology of adaptivity makes it difficult for the works in this domain to be engineered, discussed and compared. We addressed this problem by formally defining system's adaptivity, and characterising self-adaptive systems more precisely, by differentiating when a system functions and when does it adapt. However, the formal definitions are mostly declarative and descriptive and do not provide constructive insights on how to engineer self-adaptive systems, including SACPSs. To this end, we presented a logical architecture for engineering self-adaptive CPSs that narrows the gap between: (1) the MAPE-K conceptual architecture, which, over the years, has broadly served as a reference model for building self-adaptive systems, and (2) a physical architecture, i.e.,

a technical implementation for a class of decentralised SACPSs that operate in dynamic, uncertain and partially observable context. The proposed logical architecture is embedded in the formal definition of adaptivity.

The lack of consensus on *what* self-adaptive systems are, hinders the progress on *how* to specify, design and engineer these systems, and even more importantly *how* to compare and evaluate them. We hope that with the contributions of this paper, we have narrowed this gap.

CRediT authorship contribution statement

Ana Petrovska: Conceptualization, Methodology, Software, Validation, Resources, Visualization, Formal analysis, Writing – original draft. **Stefan Kugele:** Conceptualization, Methodology, Formal analysis, Supervision, Writing – review & editing. **Thomas Hutzelmann:** Conceptualization, Methodology, Formal analysis, Writing – review & editing. **Theo Beffart:** Conceptualization, Methodology. **Sebastian Bergemann:** Software, Validation. **Alexander Pretschner:** Conceptualization, Methodology, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] S. Mahdavi-Hezavehi, P. Avgeriou, D. Weyns, A classification framework of uncertainty in architecture-based self-adaptive systems with multiple quality requirements, Manag. Trade-Offs Adapt. Softw. Archit. (2017) 45–77.

[2] A. Petrovska, A. Pretschner, Learning approach for smart self-adaptive cyber-physical systems, in: 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W), IEEE, 2019, pp. 234–236.

[3] D. Weyns, Software engineering of self-adaptive systems: an organised tour and future challenges, in: Chapter in Handbook of Software Engineering, 2017.

[4] D. Weyns, T. Ahmad, Claims and evidence for architecture-based self-adaptation: a systematic literature review, in: European Conference on Software Architecture, Springer, 2013, pp. 249–265.

[5] A.J. Ramirez, A.C. Jensen, B.H. Cheng, A taxonomy of uncertainty for dynamically adaptive systems, in: 2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2012, pp. 99–108.

[6] J.O. Kephart, D.M. Chess, The vision of autonomic computing, Computer 36 (1) (2003) 41–50, http://dx.doi.org/10.1109/MC.2003.1160055.

[7] S.-W. Cheng, D. Garlan, B. Schmerl, Architecture-based self-adaptation in the presence of multiple objectives, in: Proceedings of the 2006 International Workshop on Self-Adaptation and Self-Managing Systems, 2006, pp. 2–8.

[8] D. Garlan, B. Schmerl, S.-W. Cheng, Software architecture-based self-adaptation, in: Autonomic Computing and Networking, Springer, 2009, pp. 31–55.

[9] D. Weyns, S. Malek, J. Andersson, Forms: Unifying reference model for formal specification of distributed self-adaptive systems, ACM Trans. Auton. Adapt. Syst. (TAAS) 7 (1) (2012) 1–61.

[10] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, K.M. Göschka, On patterns for decentralized control in self-adaptive systems, in: Software Engineering for Self-Adaptive Systems II, Springer, 2013, pp. 76–107.

[11] F. Quin, D. Weyns, O. Gheibi, Decentralized self-adaptive systems: A mapping study, 2021, arXiv preprint arXiv:2103.09074.

[12] J. Zhang, B.H. Cheng, Model-based development of dynamically adaptive software, in: Proceedings of the 28th International Conference on Software Engineering, 2006, pp. 371–380.

[13] A. Musil, J. Musil, D. Weyns, T. Bures, H. Muccini, M. Sharaf, Patterns for self-adaptation in cyber-physical systems, in: Multi-Disciplinary Engineering for Cyber-Physical Production Systems, Springer, 2017, pp. 331–368.

[14] F.J. Affonso, E.Y. Nakagawa, A reference architecture based on reflection for self-adaptive software, in: 2013 VII Brazilian Symposium on Software Components, Architectures and Reuse, Ieee, 2013, pp. 129–138.

[15] V. Braberman, N. D'Ippolito, J. Kramer, D. Sykes, S. Uchitel, Morph: A reference architecture for configuration and behaviour self-adaptation, in: Proceedings of the 1st International Workshop on Control Theory for Software Engineering, 2015, pp. 9–16.

[16] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, P. Steenkiste, Rainbow: Architecture-based self-adaptation with reusable infrastructure, Computer 37 (10) (2004) 46–54.

[17] R. Asadollahi, M. Salehie, L. Tahvildari, Starmx: A framework for developing self-managing Java-based systems, in: 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, IEEE, 2009, pp. 58–67.

[18] I. Gorton, Y. Liu, N. Trivedi, An extensible, lightweight architecture for adaptive J2EE applications, in: Proceedings of the 6th International Workshop on Software Engineering and Middleware, 2006, pp. 47–54.

[19] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, G.A. Papadopoulos, A development framework and methodology for self-adapting applications in ubiquitous computing environments, J. Syst. Softw. 85 (12) (2012) 2840–2859.

[20] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, U. Scholz, Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments, in: Software Engineering for Self-Adaptive Systems, Springer, 2009, pp. 164–182.

[21] J. Cámara, B. Schmerl, D. Garlan, Software architecture and task plan co-adaptation for mobile service robots, in: Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2020, pp. 125–136.

[22] C. Krupitzer, F.M. Roth, S. VanSyckel, G. Schiele, C. Becker, A survey on engineering approaches for self-adaptive systems, Pervasive Mob. Comput. 17 (2015) 184–206.

[23] I.S.C. Committee, et al., IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990), Vol. 169, IEEE Computer Society, Los Alamitos, CA, 1990.

[24] S. Kugele, A. Petrovska, I. Gerostathopoulos, Towards a taxonomy of autonomous systems, in: European Conference on Software Architecture, Springer, 2021, pp. 37–45.

[25] M. Broy, C. Leuxner, W. Sitou, B. Spanfelner, S. Winter, Formalizing the notion of adaptive system behavior, in: Proceedings of the 2009 ACM Symposium on Applied Computing (SAC), ACM, 2009, pp. 1029–1033, http://dx.doi.org/10.1145/1529282.1529508.

[26] G. Blair, N. Bencomo, R.B. France, Models@ run. time, Computer 42 (10) (2009) 22–27.

[27] A. Bennaceur, R. France, G. Tamburrelli, T. Vogel, P.J. Mosterman, W. Cazzola, F.M. Costa, A. Pierantonio, M. Tichy, M. Akşit, et al., Mechanisms for leveraging models at runtime in self-adaptive software, in: Models@ Run. Time, Springer, 2014, pp. 19–46.

[28] P.B. Kruchten, The 4+ 1 view model of architecture, IEEE Softw. 12 (6) (1995) 42–50.

[29] K. Pohl, H. Hönninger, R. Achatz, M. Broy, Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology, Springer Science & Business Media, 2012.

[30] D. Fox, Kld-sampling: Adaptive particle filters and mobile robot localization, Adv. Neural Inf. Process. Syst. (NIPS) 14 (1) (2001) 26–32.

[31] A. Petrovska, S. Quijano, I. Gerostathopoulos, A. Pretschner, Knowledge aggregation with subjective logic in multi-agent self-adaptive cyber-physical systems, in: Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2020, pp. 149–155.

[32] A. Petrovska, M. Neuss, I. Gerostathopoulos, A. Pretschner, Run-time reasoning from uncertain observations with subjective logic in multi-agent self-adaptive cyber-physical systems, in: 16th Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, 2021.

[33] A. Jøsang, Subjective Logic, Springer, 2016.

[34] N. Koenig, A. Howard, Design and use paradigms for gazebo, an open-source multi-robot simulator, in: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), IEEE, 2004, pp. 2149–2154, http://dx.doi.org/10.1109/IROS.2004.1389727.

6 Knowledge Aggregation with Subjective Logic in Multi-Agent Self-Adaptive Cyber-Physical Systems

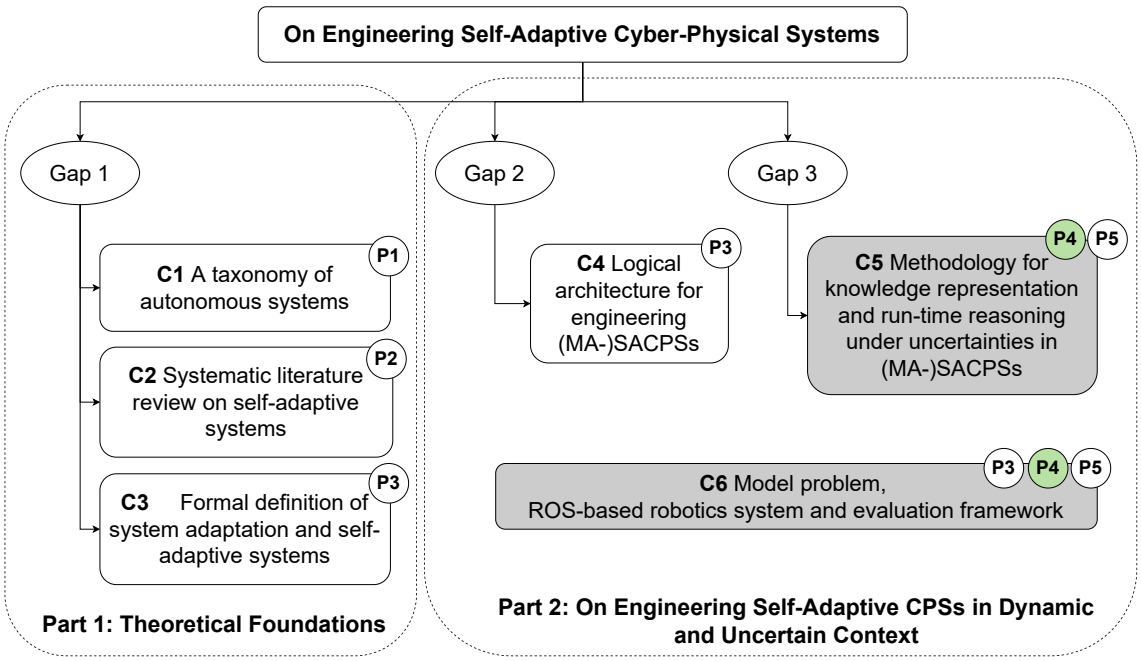


Figure 6.1: *Mapping of Gaps, Contributions and Publications to the structure of this thesis.*

Summary: The need for the *knowledge* component in the adaptation logic is broadly accepted by the majority of the works in the research domain of self-adaptive systems:

1) regardless if one assumes that self-adaptation is realised by using the traditional MAPE-K closed feedback loop or by using our logical architecture that we propose as part of the previous Chapter 5, and
2) independent from the level of abstraction and the semantic precision of the particular approach used for engineering the self-adaptive systems.

With the contributions in the previous chapters, both the theoretical foundations and the logical architecture, we have contributed to a better semantic clarity of the adaptation logic. For instance, in Chapter 5, while characterizing self-adaptive systems, we have concluded that the existence of the knowledge component in the adaptation logic, created w.r.t. the relevant aspects for the adaptation, is a minimum requirement for the system to be self-adaptive[1]. In Chapter 5 and Appendix A, we have also discussed that the knowledge

[1]Please note that this only holds in the case of passive self-adaptation, as discussed in Appendix A.

component comprises models of the system (i. e., the managed element) and the context in which the system is operating, which are relevant for the concrete system adaptation. And finally, in our logical architecture, in Chapter 5, we discussed that regardless if a self-adaptive CPSs is composed of a single or a multiple CPSs, there is a need to combine and aggregate observations from the systems (i. e., the managed elements) in order to obtain a unified, accurate and consolidated model of the context in the knowledge of the adaptation logic. Namely, since the observations made individually by each CPSs are partial and uncertain, they need to be reasoned upon and aggregated prior to updating the knowledge. However, in none of those contributions, we explicitly focused on how the knowledge in the adaptation logic can be modeled. To this end, as part of this paper, we propose a methodological approach for knowledge and uncertainty representation, which also enables reasoning under uncertainty of decentralized monitoring by the CPSs, based on which the knowledge in the adaptation logic is updated.

Problem: Due to the dynamicity of the CPSs and their execution context, the knowledge in the adaptation logic—i. e., the models of the system and the context cannot be created based on assumptions made at design time; instead, they need to be models at run-time [13, 12, 115]. For example, the run-time context model should get continuously updated in response to the changes in the context, reflecting the actual operational context during the execution of the CPSs. To update the knowledge component, the need emerges for run-time information aggregation and reasoning of what each CPSs individually observes. The observations from the CPSs can be faulty, inaccurate, partial, and potentially conflicting. In response, observation aggregation and reasoning are necessary, enabling the adaptation logic knowledge to be updated during run-time to continue reflecting the actual run-time state of the context and the system relevant for a specific adaptation.

Gap: Although knowledge representation and reasoning are important for engineering self-adaptive systems, there is a scarcity of approaches that allow domain- and system-independent modeling of the knowledge, which also allows efficiently capturing uncertain observations from one or more CPSs, based on which the knowledge is eventually updated. To our knowledge, no other work in the literature contributes to this research direction. Also, so far in the prior works, the knowledge in the adaptation logic is always constructed by the experts, and knowledge representation is treated as a system- and problem-specific task [44].

Method/Solution: We propose a methodological approach for knowledge and uncertainty representation utilizing the theoretical framework of Subjective Logic (SL) [60, 61], which supports run-time reasoning under uncertainty for a class of self-adaptive CPSs. SL is a framework for artificial reasoning, and it explicitly represents the amount of uncertainty on the degree of truth about a proposition and provides operators to combine opinions from multiple sources. We model the knowledge in the adaptation logic as a grid, in which each cell can be either occupied or not by a relevant artefact for the concrete system and the respective adaptation. A subjective opinion represents the amount of uncertainty on

the degree of truth about a proposition. For each grid cell, a subjective opinion is issued, and the subjective opinions are continuously updated during run-time based on the new information from the observations by the CPSs. Once a value, called projected probability, exceeds a certain threshold, then we have enough high certainty that the relevant artefacts (i. e., the tasks) for the CPSs actually exist in the context, then the knowledge is updated, and the CPSs can proceed with attaining the tasks. For demonstration and evaluation purposes, we used the model problem and an extension of the implementation of the robotics system, previously introduced in Chapter 5.

Results: In this paper, we conducted a preliminary evaluation, in which we compared the results when the robots attain the continuously appearing tasks in the room without any observations aggregation and with aggregation with two SL operators: CBF and CCF (see Section 2.3.3). In our results, we can observe that the time necessary to update the knowledge increases when we use knowledge aggregation, compared with the case when there is no observation aggregation (i. e., when the uncertain and potentially wrong observations are directly propagated as knowledge). This is expected since the observation and knowledge aggregation need more evidence (and hence time) to have enough certainty before eventually updating the knowledge in the adaptation logic. In response, our experiments indicated that there is a clear trade-off between the accuracy of the knowledge aggregation (i. e., how well and accurately the models in the knowledge reflect the actual state of the context) and the number of completed tasks. Our preliminary results set the foundation for additional experiments and empirical evaluations.

Contribution: As part of this paper, we did the initial conceptualization and the implementation of the methodology for knowledge and uncertainty representation in the adaptation logic, which enabled run-time reasoning of faulty, uncertain, and partial observations in self-adaptive CPSs. To this point, we showed the applicability of the theory and the formalisms of the SL in practice for the concrete research problem. Our proposed approach was evaluated using an extension of the implementation of our model problem from the robotics domain.

Limitations: In the implementation of our simulated robotics system, we aimed for the system to resemble the reality as good as possible, with the highest possible fidelity and realism, including the realistic sensing using a LiDAR sensor exactly how would a real robot sense its environment. However, while analyzing the results in more depth, we realized that the sensor model used in our initial implementation simulated an (almost too) perfect sensing that did not include the various run-time sensor uncertainties (see Section 2.1). This presented a limiting factor in our evaluation, and we address this limitation with the work in the following Chapter 7.

Author Contribution: A. Petrovska developed the initial problem statement, the reference problem from the robotics domain, as well as the design and the implementation of the robotics system. A. Petrovska and I. Gerostathopoulos jointly refined the final

version of the problem statement of this work. A. Petrovska selected the Subjective Logic as the theoretical framework and developed the theoretical solutions of the paper. S. Quijano did the implementation together with A. Petrovska. A. Petrovska, I. Gerostathopoulos and S. Quijano designed and conducted the experiments and analysed the preliminary results. The manuscript creation was mainly done by A. Petrovska in close discussion with I. Gerostathopoulos and A. Pretschner.

Knowledge Aggregation with Subjective Logic in Multi-Agent Self-Adaptive Cyber-Physical Systems

Ana Petrovska
Technical University of Munich
Munich, Germany
petrovsk@tum.de

Sergio Quijano
Technical University of Munich
Munich, Germany
sergio.quijano@in.tum.de

Ilias Gerostathopoulos
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
i.g.gerostathopoulos@vu.nl

Alexander Pretschner
Technical University of Munich
Munich, Germany
pretschn@in.tum.de

ABSTRACT

Modern software systems, such as cyber-physical systems (CPSs), operate in complex and dynamic environments. With the continuous and unanticipated change in the operational environment, these systems are subjected to a variety of uncertainties. Self-adaptive CPSs (SACPSs) can adjust their behavior or structure at run-time as a response to the changes in their perceived environment. Namely, self-adaptation is commonly realized through a MAPE-K feedback loop incorporating newly derived knowledge obtained by the sensed data from the run-time monitoring, during the operation of decentralized SACPSs. However, to build the knowledge, the need for run-time observations' aggregation and reasoning emerges, since the observations made by the decentralized systems might be conflicting. In this paper, we propose an approach for observations aggregation and knowledge modeling in SACPSs that is domain-independent and can deal with inaccurate, partial, and conflicting observations, based on the formalisms of Subjective Logic.

KEYWORDS

subjective logic, knowledge aggregation, reasoning, self-adaptive systems, cyber-physical systems

ACM Reference Format:
Ana Petrovska, Sergio Quijano, Ilias Gerostathopoulos, and Alexander Pretschner. 2020. Knowledge Aggregation with Subjective Logic in Multi-Agent Self-Adaptive Cyber-Physical Systems. In *IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '20), October 7–8, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3387939.3391600

1 INTRODUCTION

In recent years, the fast growth of cost-effective embedded systems with continuously increasing computation power has created a solid foundation for the emergence and expansion of omnipresent Cyber-Physical Systems (CPSs) across different domains, with growing socio-economic influence. CPSs are embedded systems that are distributed, networked, and interconnected. Examples range from environmental monitoring systems to robotic fleets and self-driving cars. Their close connection to the physical world means that they are exposed to high uncertainty during their operation. Namely, modern CPSs need to be able to operate efficiently and reliably within a continually changing, uncertain, and unanticipated environment (execution context) [25, 27, 29]. The context is the relevant part of the environment for a particular system. When the system under consideration is a CPS, then the relevant objects contained in the context can be other homogeneous and heterogeneous systems, entities, and processes in the physical world or cyberspace, including humans.

A common approach to deal with run-time changes and uncertainties is to make the CPSs self-adaptive. Self-adaptation is traditionally realized by an adaptation logic based on closed feedback loop—MAPE-K, with four consecutive functions, i.e., **M**onitor, **A**nalyse, **P**lan, **E**xecute with shared **K**nowledge among all the elements of the loop [11, 12, 23]. The Knowledge component comprises models of the CPSs and the context where they are operating.

Due to the dynamicity of the CPSs and their execution context, these models cannot be created based on assumptions made at design time, but instead they need to be models at run-time [5, 6, 15, 34]. Concretely, the run-time context model should get continuously updated in response to the changes in the context, reflecting the actual operational context during the execution of the CPSs. To update the model—and accoringly the Knowledge component—the need for run-time information aggregation and reasoning, of what each CPSs individually observes, emerges.

When the systems are complex and heterogeneous, like CPSs, a single MAPE loop for managing all adaptations in the system may not be sufficient [28, 36]. Instead, self-adaptive CPSs (SACPSs) typically feature more than a single MAPE loop. Having multiple control loops in SACPSs also raises a number of challenges in their development and operation. A main challenge is how to coordinate the operation of the different MAPE loops. To address this, previous works have proposed the use of design patterns for decentralized MAPE control loops in self-adaptive systems [36].

In this work, we focus on SACPSs in which MAPE-K loops are structured according to the Master-Slave pattern [36]. In this pattern, one or more MAPE-K loops at a lower level perform decentralized monitoring and execution of the adaptation actions. However, centrally, there is a single high-level MAPE-K loop that performs analysis on the monitored data, updates the knowledge, and does the planning accordingly (Figure 1). In particular, the high-level MAPE-K loop needs to reason on the uncertain, inaccurate, and potentially conflicting observations coming from the decentralized monitoring, which, once aggregated, they become knowledge.

SEAMS '20, October 7–8, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7962-5/20/05…$15.00
https://doi.org/10.1145/3387939.3391600

Knowledge aggregation in SACPSs that follows the Master-Slave pattern or any other pattern with decentralized monitoring and centralized analysis is essential since it can have a significant influence on the subsequent phases of planning and executing. Consider, for example, the case of a route planning algorithm that relies on knowledge aggregation from distributed sensors from multiple agents, which may give conflicting readings. Knowledge aggregation in SACPSs is also challenging, since, as mentioned above, it needs to take into account the inherent uncertainties related to each monitor and provide ways to synthesize and consolidate knowledge in different cases where conflicts arise.

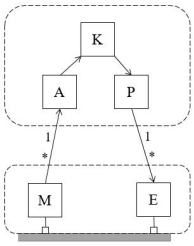


Figure 1: Modified Master-Slave pattern from [36].

Although knowledge and its aggregation is an important concern for MAPE-K patterns as also acknowledged by Weyns et al. [36], there is a scarcity of approaches for modeling self-adaptation knowledge that allows for capturing uncertainty at a local level and for effectively aggregating knowledge for decision making at a global level. Instead, knowledge modeling is typically treated as a domain-specific task, leading to ad-hoc solutions to knowledge aggregation.

In response, in this paper, we present an approach for observations aggregation and knowledge modeling in SACPSs that is domain-independent and can deal with reasoning on inaccurate, partial, and conflicting observations. Concretely, our approach uses Subjective Logic to build knowledge by aggregating partial observations of the context made by each agent in a decentralized multi-agent SACPSs.

Subjective Logic is an enriched probabilistic logic-based framework for artificial reasoning, based on which Subjective Opinions about a knowledge item from the different monitors are created. Additionally, Subjective Logic proposes different fusion operators that allow aggregating the opinions to a final actionable set of knowledge items in the analysis phase, based on which the runtime context model in the Knowledge component is accordingly updated, and later utilized to plan the next adaptation actions.

In short, the main contributions of the paper are the following:

- A subjective logic-based approach for knowledge aggregation in decentralized monitoring of partial context observation in SACPSs.
- An open-source implementation of a ROS-based simulated multi-robot system, based on the robotics use case, further explained in Section 2.

2 RUNNING EXAMPLE

To motivate the need for knowledge aggregation in self-adaptive systems and illustrate our approach, we introduce a reference problem from the domain of CPSs, in particular from the robotics domain, which is also used as a running example throughout the paper.

The reference problem is comprised of several cleaning robots operating in the same context, e.g., a room. Each robot is able to autonomously move to a destination while avoiding 1) static obstacles (e.g., walls, furniture, etc.) and 2) dynamic obstacles (e.g., other robots, humans) along its way. New dirt tasks continuously appear in the room, and the robots discover them in a distributed manner with a 2D laser LIDAR scanner, capable of sensing 360 degrees radius. The robot's observation space is determined by the scanning distance of the laser scanner, which is less than the room's dimensions. Hence, each robot can only observe partially the room in which it operates. Consequently, the robots can detect the newly appearing dirt tasks only if they are within their range of observation.

The mission of the robots is to keep the room as clean as possible by discovering the dirt tasks and then cleaning them most efficiently. However, the fact that the robots have only a partial observation of the room brings inefficiency to the overall performance, for example, when one part of the room is getting dirtier than the other.

Therefore, the partial observations made by each robot are sent periodically to a `Cleaning Controller`, whose responsibility is to aggregate the received observations and assign the discovered dirt tasks to the robots, while respecting the optimality criteria. Once a dirt task is assigned to the robot, it navigates and moves to the location of the corresponding task, accomplishes it, and then navigates to the next task in its queue.

We model the context as a grid map with a size equal to the size of the room (Figure 3). The cells in the grid are either occupied by static obstacles, or by dynamic obstacles: robots or dirt tasks. A context variable models whether a specific cell of the grid map has a dirt task or not, so there are as many context variables as cells in the map of the room. Dirt tasks appear per cell; therefore, in the paper, we use dirt task and dirt cell interchangeably.

The multi-robot cleaning system is subject to external (contextual) and internal (system) uncertainties, manifested via the continuous appearance of tasks in the room and different sensor uncertainties, respectively. Occasionally, each robot will mistakenly sense a dirt task when there is none, and on the contrary, fail to sense an actual task. This potentially results in different robots holding different opinions regarding the space they observe, which requires appropriate conflict resolution during the aggregation process.

In this setting, each robot is an agent of the SACPS and can be modeled via a low-level MAPE-K loop (with respect to the Master-Slave pattern). Each robot independently *monitors* its surroundings and *executes* actions to accomplish its assigned tasks. `Cleaning Controller` has the role of the high-level MAPE-K loop performing the centralized *analysis* and *planning*.

3 BACKGROUND ON SUBJECTIVE LOGIC

Subjective Logic (SL) [19, 20] is a framework for artificial reasoning, in which the general idea is to enrich probabilistic logic by explicitly including (1) uncertainty about probabilities and (2) subjective belief ownership. It allows to express a degree of (un)certainty about a subjective belief (called *opinion*).

To reason with propositions whose truth values are uncertain, Bayesian probability and statistics can also be employed [18]. However, this type of probabilistic logic does not allow to seamlessly model situations where different agents express their beliefs about

the same proposition. SL explicitly integrates the subjective nature and ownership of beliefs in its formalism, allowing the combination of different beliefs about the same proposition. Nevertheless, the interpretation of a SL opinion in the Bayesian perspective is possible by mapping opinions into probability distributions [19].

SL is based on the Dempster-Shafer Theory of evidence (DST), a flexible theoretical framework to represent uncertainty introduced by Dempster [13] and Shafer [32]. In particular, DST's rule of combination, originally proposed for merging sources of evidence in DST, is also used in SL where it represents a method of preference combination embodied in SL's *belief constraint operator* [21]. Moreover, the idea of explicit representation of ignorance is inherited from the Dempster-Shafer belief theory [19, 32].

3.1 Subjective Logic Opinions

The fundamental building block of SL is a *subjective opinion* that represents the amount of uncertainty on the degree of truth about a proposition. The representation of a subjective opinion is a composite function consisting of belief masses, uncertainty mass and base rate. An opinion expresses a belief about the state of a variable which takes its values from a domain (t.e., a state space).

A domain represents all the possible states of a variable situation. Domains can be binary or n-ary. A binary domain can be denoted $\mathbb{X} = \{x, \overline{x}\}$, where $\overline{x}$ is the complement of x. Binary domains are typically used when modeling situations that have only two alternatives, such as the case of our running example, where a location in the map can either have dirt or not. Situations with more than two alternatives have n-ary domains. If $\mathbb{X}$ denotes a binary or an n-ary domain, X can be a random variable which takes its values from $\mathbb{X}$. For instance, we model a situation in our running example using binary random variables X that take their values from the binary domain $\mathbb{X} = \{dirt, no\_dirt\}$.

Binomial Opinion Representation. In SL, the notation w_X^A is used to denote opinions, where X indicates the target variable or proposition to which the opinion applies, and A indicates the agent who holds the opinion. Opinions on binomial variables (e.g. variables with domain $\mathbb{X} = \{x, \overline{x}\}$) are called binomial opinions, and a special notation is used for their mathematical representation.

Definition 1 (Binomial Opinion [20]). Let $\mathbb{X} = \{x, \overline{x}\}$ be a binary domain with binomial random variable $X \in \mathbb{X}$. A binomial opinion about the truth/presence of value x is the ordered quadruplet $w_x = (b_x, d_x, u_x, a_x)$, where the additivity requirement $b_x + d_x + u_x = 1$ is satisfied, and where the respective parameters are defined as

b_x: belief mass in support of x being TRUE (i.e. $X = x$),
d_x: disbelief mass in support of x being FALSE (i.e. $X = \overline{x}$),
u_x: uncertainty mass representing the vacuity of evidence,
a_x: base rate, i.e., prior probability of x without any evidence.

Opinions with $u_x = 1$ and $u_x = 0$ are called *vacuous* and *dogmatic*, respectively. Finally, the expected probability of a binomial opinion about value x is defined by: $P(x) = b_x + a_x u_x$

Illustration on the Running Example. In our running example, each robot R issues an opinion for each cell (i, j) that they are able to observe. A binomial opinion about the presence of dirt on a cell is the ordered quadruplet $w_{i,j}^R = (b_x, d_x, u_x, a_x)$ with

b_x: belief mass in support of a tile being dirty,
d_x: disbelief mass in support of no dirt,
u_x: uncertainty of the sensor observation,
a_x: 1/2 (taking an unbiased viewpoint).

The belief mass distribution is calculated as a function of the robot's sensor range and the distance to a detected object. In Section 4 we discuss the details of the detection process and the corresponding belief/disbelief and uncertainty masses calculation.

3.2 Combination of Subjective Logic Opinions

When there are more than one opinions for a proposition, there is often the need to merge or combine them into a single collective opinion. Such knowledge aggregation with Subjective Logic can be realized through a process called *Belief fusion* [20]. Multiple distinct agents, denoted A_1, A_2, ... A_N, can produce different and possibly conflicting opinions $w_X^{A_1}$, $w_X^{A_2}$,...$w_X^{A_N}$ about the same variable X. Multi-source fusion consists of merging the different sources into a single source that can be denoted $\diamond(A_1, A_2, ..., A_N)$, and mathematically fusing their opinions into a single opinion denoted $w_X^{\diamond(A_1, A_2, \ldots, A_N)}$.

Subjective logic provides a variety of operators, generalizing and extending operators from binary logic and probability calculus, including different belief fusion operators: *averaging belief fusion, cumulative belief fusion, weighted belief fusion, consensus & compromise fusion*, and *belief constraint fusion* [33]. Each of these fusion operations is designed to determine the shared belief and uncertainty of a group of evidence sources, with different applications depending on how evidence should be combined.

In the rest of the section, we detail on the Cumulative Belief Fusion and Consensus & Compromise Fusion operators, which we have experimented with in the running example.

Cumulative Belief Fusion (CBF). CBF is suitable when it is assumed that the amount of independent evidence increases with the inclusion of more independent sources [20, 22]. If no dogmatic opinion is present, CBF cumulates the evidence parameters of all opinions. Alternatively, only dogmatic opinions are considered in the cumulation of evidence. Vacuous opinions have no influence on the result. Applying CBF to non-conflicting, uncertain opinions reduces the uncertainty of the resulting opinion. On the other hand, applying CBF to conflicting opinions with the same uncertainty mass has the effect of canceling them out. The CBF operator is associative, commutative, and non-idempotent. The last property means that the fusion of equal opinions will in general produce an opinion that is different from the initial ones.

Consensus & Compromise Fusion (CCF). CCF is suitable when there is a need for keeping shared beliefs from different sources and transforming conflicting beliefs into compromise belief [20]. Conflict resolution is achieved by first computing a consensus, conserving the agreed weight of all sources, and then computing a weighted compromise for the residue belief mass based on the relative uncertainty and the corresponding base rates [33]. Similar to CBF, vacuous opinions are neutral elements in CCF fusion. Contrary to CBF though, CCF is idempotent, meaning that fusing equal opinions produces the same opinion. Technically, the calculation of CCF consists of three phases, namely the consensus phase, the compromise phase, and the normalization phase[22, 33].

Illustration on the Running Example. The different robots in our running example are the agents that hold independent opinions

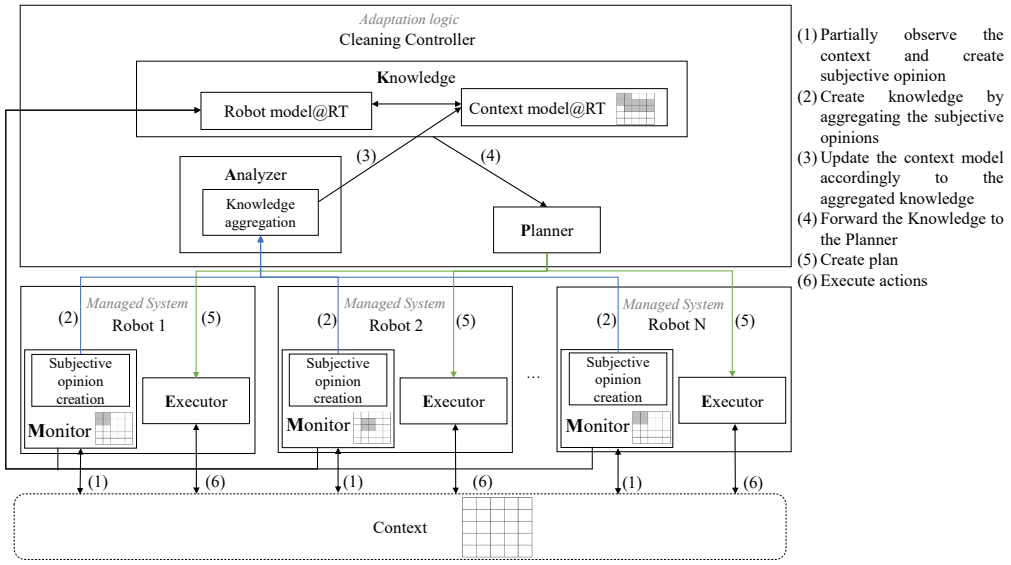


Figure 2: Overview of the approach.

about the cells that they observe being dirty or not. CBF is known to be well suited for fusing opinions coming from sensor-generated evidence [22], while CCF is well suited for fusing opinions coming from different experts. Both operators can be used in the running example, since they can cope with shared beliefs (e.g. robots detecting dirt in a cell with low confidence), conflicting beliefs (e.g. when a robot detects dirt in a cell due to sensor noise while another does not), and vacuous opinions (e.g. a robot does not observe a cell at all since it is out of its LIDAR range). Choosing CCF over CBF allows us to better deal with totally conflicting opinions at the cost of higher uncertainty in the merged opinion.

4 APPROACH

4.1 Overview of the Approach

Our approach assumes that there are several MAPE-K loops in the adaptation logic of the SACPS under study and that the MAPE-K loops are structured accordingly to the Master-Slave pattern [36]. In particular, there are several decentralized Monitor and Execute components, and single centralized Analysis and Plan components, as depicted in the overview of the approach in Figure 2.

Monitor. Monitor components belong to the lower-level MAPE-K loops. They make independent observations about the SACPS itself and the context in which the SACPSs operate. These observations are partial—only cover part of the context, and incorrect—they may include mistakes due to sensor noise and inaccuracies. Each Monitor has a Subjective Opinion Creator entity, which is responsible for creating Subjective Logic opinions from each agent about different context variables. Once the Subjective Logic opinions are created, they are independently forwarded to the Analysis component. The Subjective Opinion Creator is further explained in Section 4.2.

Running example. Each robot in our running example represents a monitor component. It periodically senses its own position and the presence of dirt tasks in the room. Its observations are both partial, due to limited range of its LIDAR sensors, and sometimes incorrect, due to noise in its LIDAR sensors. For illustration, Figure 3 shows that each robot, at a point in time, can only observe part of the context.

Analysis. The analyzer is a centralized component in our approach that collects the Subjective Logic Opinions from the distributed Monitor components and updates the run-time context model in the Knowledge component. In particular, the Knowledge Aggregator entity is responsible for combining the opinions from different agents about different context variables using a Subjective Logic operator.

Running example. In our running example, the Analyzer is a component housed in `Cleaning Controller`. It aggregates the partial observations by fusing the opinions made by robots for the cells that they are observing. The Knowledge Aggregator is further explained in Section 4.3.

Plan. The Planner is another centralized component in our approach that is responsible for selecting adaptation actions or plans, which are later being executed by each lower-level MAPE-K loop. We assume that the Planner relies on the run-time context model represented by the context variables, upon which agents issue opinions in the previous step. However, we do not prescribe how to perform planning: any planning approach (e.g., rule-based, goal-oriented) can be used within our approach.

Running example. In our running example, the Planer is also part of the centralized `Cleaning Controller`. It takes as input the aggregated knowledge in terms of fused opinions about the appearance of dirt tasks in the cells. The discovered, unassigned tasks are assigned to the robots, as exlained in Section 2.

Execute. Executor components belong to the lower-level MAPE-K loops. They obtain adaptation actions or plans from the Planner of the higher-level MAPE-K loop and independently execute them.

Running example. In the running example, each robot represents an Executor component. It keeps a self-adaptive priority queue of the locations of the dirt tasks that a robot needs to accomplish. The queues of the robots are modified at run-time, based on the distance of the robot closest to the newly appeared cleaning task. As long

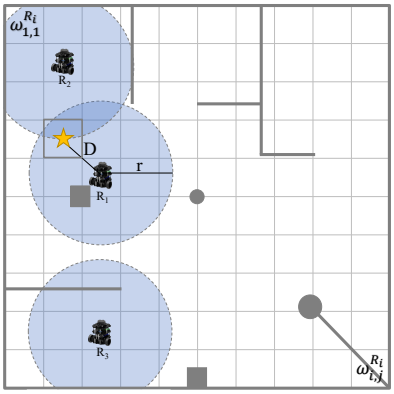


Figure 3: Grid map and partial context observation

as there are tasks in its queue, each robot picks the next task and navigates autonomously to the corresponding cell to clean the task.

4.2 Subjective Opinion Creator

A robot R issues a subjective opinion $w^R_{i,j}$ for each cell (i, j) in the grid map that it is within its detection range r (Figure 3). This opinion depends on the Euclidean distance D of the cell from the robot and models whether that cell contains dirt or not. In particular, the parameters of an opinion $w^R_{i,j} = (b_X, d_X, u_X, a_X)$ are calculated by the following formulas:

$$b_X = \begin{cases} 1 - u_X, \text{for } X = dirt \\ 0.0, otherwise \end{cases} \qquad d_X = \begin{cases} 1 - u_X, \text{for } X = no\_dirt \\ 0.0, otherwise \end{cases}$$

$$u_X = min(0.99, \frac{D}{r}) \qquad a_X = 1/2$$

The further a cell is from the robot, the higher the uncertainty mass u_X of the robot's opinion. Observations at the edge of LIDAR sensor range are considered highly uncertain, but can still contribute during knowledge aggregation; hence, we assign an uncertainty value of 0.99 instead of a totally uncertain opinion (i.e., $u_X = 1$). If a robot detects dirt on a cell, the belief mass b_X becomes the complement of u_X and the disbelief mass d_X for that cell becomes zero. On the contrary, if no dirt is detected, b_X becomes zero and d_X becomes the complement of u_X. The base rate a_X is always considered to be the default base rate for a binary domain, i.e. $a_X = 1/2$. Finally, no subjective opinions are issued for (i) cells which are occupied by static obstacles (e.g. walls) since these are assumed to be accurately detected, (ii) cells that lie outside of the detection range of the robot.

4.3 Knowledge Aggregator

When the system is first initialized, the run-time context model does not contain information that the Analyzer can use to carry out its tasks. We model this situation by creating a vacuous subjective opinion for each cell of the context grid map. This initialization serves two purposes in our subjective logic approach: (1) a vacuous opinion will inform the analyzer that there is no previous knowledge about a context variable and (2) when the first knowledge aggregation is executed, the existing vacuous opinions do not influence in the final result.

Opinion	No conflict				Conflicting observations			
	Inputs		Aggregation		Inputs		Aggregation	
	R_1	R_2	CBF	CCF	R_1	R_2	CBF	CCF
b_x	0.630	0.010	0.631	0.634	0.000	0.010	0.004	0.004
d_x	0.000	0.000	0.000	0.000	0.630	0.000	0.628	0.630
u_x	0.370	0.990	0.369	0.366	0.370	0.990	0.369	0.366
a_x	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500
$P(x)$	0.815	0.505	0.816	0.817	0.185	0.505	0.188	0.187

Table 1: Knowledge aggregation of two robots' observations

The knowledge aggregation takes place each time a robot informs the Knowledge Aggregator of a made partial observation of the context. For the cells covered in the partial observation, subjective opinions are issued. Additionally, the Knowledge Aggregator extracts the previously-stored opinions for the corresponding cells from the Knowledge. For each cell, both opinions are then fused and aggregated, and the run-time context model in the Knowledge is updated accordingly. This process is executed with the same frequency for all the robots scanning their surrounding area.

This aggregation step aims to solve potential conflicting observations, and at the same time, increase the confidence of the observations, according to the chosen subjective logic operator.

As an example, let us consider the situation illustrated in Figure 3, where robots R_1 and R_2 hold overlapping partially observed contexts and a dirt cell (depicted with a yellow star) is detected. When the knowledge aggregator receives the independent partial observations sent by the robots, it initiates the aggregation process. Let us further consider two different scenarios: (1) R_1 and R_2 detect dirt in the same cell and, (2) R_1 does not detect dirt but R_2 does. Using the formulas described in Section 4.2, Table 1 shows the calculated opinions for both scenarios, and the knowledge aggregation results using CBF and CCF operators.

In both scenarios, we observe improvement in the confidence of the aggregated observations, i.e., the uncertainty mass in the aggregated opinion decrease compared to the individual uncertainty masses of each robot's opinion (blue cells in Table 1). However, in the presence of conflicting observations, the resulting belief masses reflect a consensus and compromise of the individual robots' opinions (yellow cells in Table 1); this compromise belief is reflected in the expected probability $P(x)$ calculated as explain in Section 3.1. The resulting $P(x)$ of the aggregated opinions is then used by the Analyzer to decide when the detected dirt should be considered as a goal to be assigned to one of the participating robots.

5 IMPLEMENTATION

In this section, we discuss the implementation of our testbed based on the reference problem, which was previously described in Section 2. Robotics is an inherently complex domain, particularly when considering not only one but multiple agents. So merely creating and setting up a realistic multi-robot system presents a challenge by itself. By providing a *simulated*, yet physically correct representation of the robots with their sensors and actuators, and the context where they are operating, our current implementation provides a foundation for various applications and experiments, which can also be easily modified accordingly to the individual needs of other researchers.

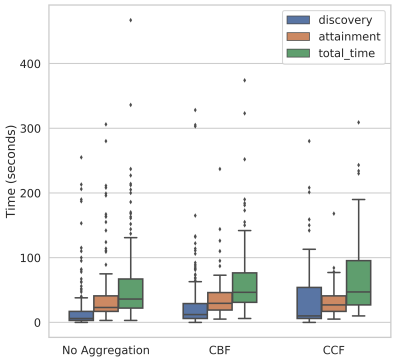


Figure 4: Result for a simulation of 2 robots with and without knowledge aggregation.

In our implementation, the entire communication is based on the Robot Operating System (ROS). For simulating the robotics system we use Gazebo [1, 24], and the robots we simulate are *TurtleBot 3 Burger*[1]. With our implementation, one can simulate as many robots, as long the number of robots makes sense in the given room. Additionally, different room maps, from the one currently considered in our system, can be used. Gazebo relies on a well-established physics engines, which enables high physical [10], functional [2] and visual [31] fidelity. Simulations with high fidelity closely resemble the real world, meaning that the model and even the physics of the robots, the environment, including the static and dynamic objects are simulated as realistically as possible. The use of ROS allows the system to be deployed on real, physical robots without any modification. The source code of the implementation, together with complete documentation, and installation instructions are available on the following link: https://github.com/squijanor/knowledge-aggregation-subjective-logic.git. For subjective logic, we have used an open-source Java library[2].

6 PRELIMINARY EXPERIMENTS

6.1 Setup

The testbed provides a simulation of n robots deployed in a room spanning 10 x 10m. To evaluate the knowledge aggregation approaches, we conducted five series of experiments, 10 minutes each. For the simulation of the appearance of new tasks, we used a random seed in each experiment, and to guarantee a better replication of the test scenarios, we used a fixed frequency with which the new tasks are created. We compare these results with a base scenario without knowledge aggregation. In this scenario, the `Cleaning Controller` does not aggregate nor solve conflicts in the observations made by different robots; instead, it proceeds to create goals directly from the observations received from every individual robot. This might result in goals at locations where dirt tasks do not exist or locations that are different from the real locations of the tasks.

6.2 Preliminary Results

Figure 4 shows first results from experiments comparing no aggregation, and aggregation with CBF and CCF. CBF requires the fusion of different sources to increase the confidence in the observations, whereas CCF trades-off conflict resolution for a higher uncertainty in the merged result. We used 0.5 as expected probability threshold in the aggregated subjective opinions to determine when the detected task should be assigned as a goal to a robot. By adjusting this parameter, the effect on the discovery time can be controlled. In our first experimental results, we see that the discovery time increases for both CBF and CCF. This is expected, since they need more evidence (and hence time) to create cleaning goals. The attainment time is roughly the same in all cases.

7 RELATED WORK

Models@Runtime Models@Runtime [5, 6, 15, 34] are a promising approach to managing complexity in run-time environments, based on software models. They are considered as adaptation mechanisms, or rather support for realizing self-adaptive systems. According to [6] the run-time models provide "abstractions of run-time phenomena" and they can be used in a various ways by different stakeholders. Additionally, the models should represent the system by reflecting the system and its current state and behavior. Namely, if the underlying system changes, then the representations of the system—the models—should also change. Floch et al. in [15] and Bennaceur et al. in [5] identify the need for mechanisms to reason about the system and its environment in models@runtime, without proposing any concrete solution.

Uncertainties in Self-adaptive systems. The term uncertainty has been broadly discussed across many disciplines and sciences. However, in the field of self-adaptive systems, uncertainties have a central role as the main triggers for a system to self-adapt so that the business continuity of the system can be preserved during run-time. Across the literature, there have been many proposed works that have tried to understand the scope and the effects that the uncertainties have on the dynamic systems[8, 14, 16, 25, 26, 30, 35]. Ramirez et al. [30] classify uncertainties on three different levels, based on the uncertainties' sources: uncertainties on the requirement level, design level and runtime level. According to the authors, the potential sources of runtime uncertainties are mainly related to interactions between the system and its context, including sensor noise, inaccuracy of sensor measurements, or an unpredictable system environment. Besides the proposed mitigation techniques for unpredictable environment [3, 7, 9, 17, 37], sensor failure [7, 17] and incomplete information [4, 9, 37], there are no identified techniques to mitigate the rest of the run-time uncertainties including sensor noise, sensor imprecision and inconsistency [30]. We hope that with the proposed approach in this paper, we are contributing towards narrowing this gap.

8 CONCLUSION

In this paper, we propose an approach that uses subjective logic to collaboratively aggregate the partial observations of the context made by each CPS (robot) in a multi-agent setup. Based on the aggregated observations, we accordingly update the run-time context model of the context in the knowledge of the adaptation logic, which is later utilized by the adaptation logic to analyze and plan the next adaptation actions.

[1] https://www.turtlebot.com/
[2] https://github.com/vs-uulm/subjective-logic-java

Through preliminary experiments, we demonstrated that Subjective logic is a flexible framework to merge knowledge from different agents. The provided testbed sets the basis to experiment with different methods for knowledge aggregation. In this work we only evaluated the application of CBF and CCF operators. In future work, we plan to evaluate the remaining set of operators, which will provide a complete insight into the full capabilities of subjective logic as a means for knowledge aggregation. Also, to guarantee the internal validity of our testbed, we will conduct different and multiple trials to determine the statistical soundness of our results. Namely, we plan to increase the number of experiments, testing different room distributions and the simulation of multiple robots.

ACKNOWLEDGMENTS

This research has in part been funded by the *German Federal Ministry of Education and Research* under the grants no. 01IS16043A.

REFERENCES

[1] Carlos E. Aguero, Nate Koenig, Ian Chen, Hugo Boyer, Steven Peters, John Hsu, Brian Gerkey, Steffi Paepcke, Jose L. Rivero, Justin Manzo, Eric Krotkov, and Gill Pratt. 2015. Inside the Virtual Robotics Challenge. 12 (2015), 494–506. Issue 2. https://doi.org/10.1109/TASE.2014.2368997

[2] John A. Allen, Robert T. Hays, and Louis C. Buffardi. 1986. Maintenance Training Simulator Fidelity and Individual Differences in Transfer of Training. 28 (1986), 497–509. Issue 5. https://doi.org/10.1177/001872088602800501

[3] Luciano Baresi, Liliana Pasquale, and Paola Spoletini. 2010. Fuzzy goals for requirements-driven adaptation. In *2010 18th IEEE International Requirements Engineering Conference*. IEEE, 125–134.

[4] Nelly Bencomo, Jon Whittle, Pete Sawyer, Anthony Finkelstein, and Emmanuel Letier. 2010. Requirements reflection: requirements as runtime entities. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, Vol. 2. IEEE, 199–202.

[5] Amel Bennaceur, Robert France, Giordano Tamburrelli, Thomas Vogel, Pieter J Mosterman, Walter Cazzola, Fabio M Costa, Alfonso Pierantonio, Matthias Tichy, Mehmet Akşit, et al. 2014. Mechanisms for leveraging models at runtime in self-adaptive software. In *Models@ run. time*. Springer, 19–46.

[6] Gordon Blair, Nelly Bencomo, and Robert B France. 2009. Models@ run. time. *Computer* 42, 10 (2009), 22–27.

[7] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. 2009. Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*. Springer, 48–70.

[8] Javier Cámara, David Garlan, Won Gu Kang, Wenxin Peng, and Bradley Schmerl. 2017. Uncertainty in Self-Adaptive Systems Categories, Management, and Perspectives. (2017).

[9] Betty HC Cheng, Pete Sawyer, Nelly Bencomo, and Jon Whittle. 2009. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 468–483.

[10] Jeff Craighead, Robin Murphy, Jenny Burke, and Brian Goldiez. 2007. A Survey of Commercial & Open Source Unmanned Vehicle Simulators. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation*. IEEE, 852–857. https://doi.org/10.1109/ROBOT.2007.363092

[11] Rogério De Lemos, David Garlan, Carlo Ghezzi, Holger Giese, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Danny Weyns, Luciano Baresi, Nelly Bencomo, et al. 2017. Software engineering for self-adaptive systems: Research challenges in the provision of assurances. In *Software Engineering for Self-Adaptive Systems III. Assurances*. Springer, 3–30.

[12] Rogério De Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. 2013. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*. Springer, 1–32.

[13] A. P. DEMPSTER. 1968. A Generalization of Bayesian Inference Author (s): A . P . Dempster Source : Journal of the Royal Statistical Society . Series B (Methodological), Vol . 30 , No . 2 Published by : Wiley for the Royal Statistical Society Stable URL : http://www.jstor.o. 30, 2 (1968), 205–247.

[14] Naeem Esfahani and Sam Malek. 2013. Uncertainty in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*. Springer, 214–238.

[15] Jacqueline Floch, Svein Hallsteinsen, Erlend Stav, Frank Eliassen, Ketil Lund, and Eli Gjorven. 2006. Using architecture models for runtime adaptability. *IEEE software* 23, 2 (2006), 62–70.

[16] David Garlan. 2010. Software engineering in an uncertain world. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 125–128.

[17] David Garlan, S-W Cheng, A-C Huang, Bradley Schmerl, and Peter Steenkiste. 2004. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer* 37, 10 (2004), 46–54.

[18] Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. 2014. Part I Fundamentals of Bayesian Inference. In *Bayesian Data Analysis* (3. ed. ed.). CRC Press, Chapter 1, 4–29.

[19] Audun Jøsang. 2001. A Logic for Uncertain Probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 9, 3 (2001), 271–311.

[20] Audun Jøsang. 2016. *Subjective Logic*. Springer International Publishing Switzerland. https://doi.org/10.1007/978-3-319-42337-1

[21] Audun Jøsang and Simon Pope. 2012. Dempster's rule as seen by little colored balls. *Computational Intelligence* 28, 4 (2012), 453–474. https://doi.org/10.1111/j.1467-8640.2012.00421.x

[22] Audun Josang, Dongxia Wang, and Jie Zhang. 2017. Multi-source fusion in subjective logic. *20th International Conference on Information Fusion, Fusion 2017 - Proceedings* (2017). https://doi.org/10.23919/ICIF.2017.8009820

[23] Jeffrey O. Kephart and David M. Chess. 2003. The Vision of Autonomic Computing. *Computer* 36, January (2003), 41–50. https://doi.org/10.1046/j.1365-2745.2002.00730.x arXiv:arXiv:astro-ph/0005074v1

[24] Nathan Koenig and Andrew Howard. 2004. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. IEEE, 2149–2154. https://doi.org/10.1109/IROS.2004.1389727

[25] Sara Mahdavi-Hezavehi, Paris Avgeriou, and Danny Weyns. 2017. A classification framework of uncertainty in architecture-based self-adaptive systems with multiple quality requirements. In *Managing Trade-Offs in Adaptable Software Architectures*. Elsevier, 45–77.

[26] Diego Perez-Palacin and Raffaela Mirandola. 2014. Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation. In *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*. ACM, 3–14.

[27] Ana Petrovska and Alexander Pretschner. 2019. Learning Approach for Smart Self-Adaptive Cyber-Physical Systems. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE, 234–236.

[28] Mariachiara Puviani, Giacomo Cabri, and Franco Zambonelli. 2013. A taxonomy of architectural patterns for self-adaptive systems. In *Proceedings of the International C* Conference on Computer Science and Software Engineering*. ACM, 77–85.

[29] Federico Quin, Thomas Bamelis, Singh Buttar Sarpreet, and Sam Michiels. 2019. Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 1–12.

[30] Andres J Ramirez, Adam C Jensen, and Betty HC Cheng. 2012. A taxonomy of uncertainty for dynamically adaptive systems. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 99–108.

[31] Edward J. Rinalducci. 1996. Characteristics of Visual Fidelity in the Virtual Environment. 5 (1996), 330–345. Issue 3. https://doi.org/10.1162/pres.1996.5.3.330

[32] Glenn Shafer. 1976. A mathematical theory of evidence. *Princeton University Press* (1976).

[33] Rens W. Van Der Heijden, Henning Kopp, and Frank Kargl. 2018. Multi-Source Fusion Operations in Subjective Logic. *2018 21st International Conference on Information Fusion, FUSION 2018* (2018), 1990–1997. https://doi.org/10.23919/ICIF.2018.8455615 arXiv:arXiv:1805.01388v1

[34] Thomas Vogel, Andreas Seibel, and Holger Giese. 2010. The role of models and megamodels at runtime. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 224–238.

[35] Kristopher Welsh and Pete Sawyer. 2010. Understanding the scope of uncertainty in dynamically adaptive systems. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2–16.

[36] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M. GÃűschka. 2013. On Patterns for Decentralized Control in Self-Adaptive Systems. In *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, RogÃŕrio de Lemos, Holger Giese, Hausi A. MÃijller, and Mary Shaw (Eds.). Springer, Berlin, Heidelberg, 76–107. https://doi.org/10.1007/978-3-642-35813-5_4

[37] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty HC Cheng, and Jean-Michel Bruel. 2009. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *2009 17th IEEE International Requirements Engineering Conference*. IEEE, 79–88.

7 Run-time Reasoning from Uncertain Observations with Subjective Logic in Multi-Agent Self-Adaptive Cyber-Physical Systems

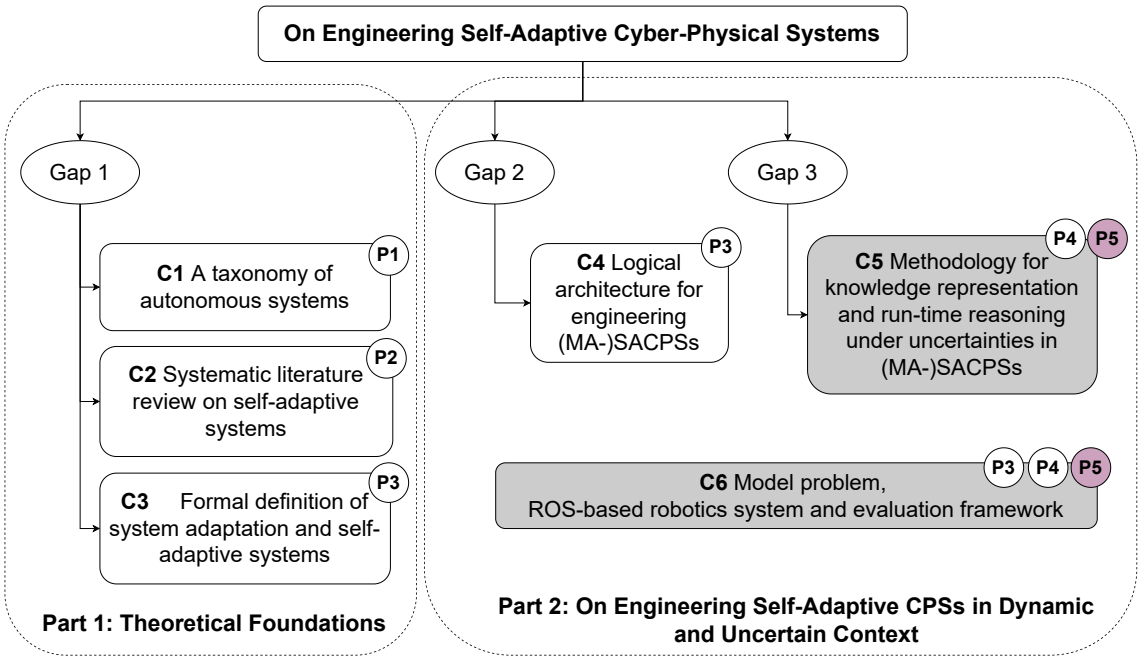


Figure 7.1: *Mapping of Gaps, Contributions and Publications to the structure of this thesis.*

Summary: In this section, we build on our previous work from Chapter 6, in which we introduced 1) the idea of knowledge aggregation and reasoning with SL, 2) the conceptualization of the methodology for knowledge and uncertainties representation in self-adaptive systems and 3) the preliminary results. Concretely, in this paper, we further detail and develop our methodology and the implementation of the robotics system and extensively evaluate our fully developed approach. Hence, we address the limitations from Chapter 6, and some of the limitations of the SL that we discovered during the initial evaluations while working on this paper. The problem and the gap that this paper contributes to remain the same as in Chapter 6.

Method/Solution: As part of the paper, we propose two additional solutions that contribute towards the fully developed SL-based methodology for knowledge and uncertainty representation and reasoning under uncertainties in self-adaptive CPSs. The proposed

solutions in this paper extend the methodological and technical solutions that we previously proposed in Chapter 6. The first solution in this paper is a technical solution, in which we extend the implementation of the robotics system, concretely the implementation of the sensing, to support various run-time uncertainties as discussed in Section 2.1. Additionally, our results from the initial evaluation of the fully developed robotics system showed that none of the original operators for information fusion originally proposed by the SL supports (cf. Section 2.3.3) a long-term knowledge aggregation. In response, as a second solution, we propose a new SL fusion operator, which we named *Combination Scheme*.

Results: We identify three research questions that guide our evaluation. In the first research question, we investigate the feasibility of different SL fusion operators (i. e., aggregation schemes) for knowledge aggregation. We initially aimed to evaluate our proposed approach using either CBF or CCF as SL fusion operators. However, our initial evaluation, conducted as part of this paper, showed that none of the SL operators is capable of providing long-term, real-time knowledge aggregation. To show this, we conducted analytical and empirical evaluations. In the second research question, we evaluate the effectiveness of the best aggregation scheme on the core research problem. In other words, mitigating uncertainties from various uncertain, faulty, and potentially conflicting observations by the CPSs based on which the knowledge in the adaptation logic of the self-adaptive CPSs is modified and updated during run-time. Our results have shown that our approach is capable of correcting the observations of a faulty CPS. And finally, in the third research question, we investigate the sensitivity of the approach to the main parameter for deciding when the knowledge in the adaptation logic is updated—the expected probability of the context variables. With the results from our evaluation, we validated the indication from the preliminary results from Chapter 6 and showed a clear trade-off between the number of tasks completed and the accuracy of the CPSs.

Contribution: In this work, we propose a fully developed domain-independent methodological approach for knowledge and uncertainty representation and knowledge aggregation and reasoning of decentralized monitoring in (MA-)SACPSs that inherently produce uncertain, faulty, and potentially conflicting observations. Our proposed methodology allows to efficiently capture and mitigate uncertainty at run-time, based on which the knowledge (i. e., the models) in the adaptation logic is updated during run-time to accurately reflect the actual state of the context and the CPSs during the operation of the self-adaptive CPSs. Our methodological contribution is additionally supported by a technical contribution from the robotics domain. Concretely, an open source implementation of an in-house ROS-based multi-robot system, which as part of this paper acted as a testbed for experimentation with all the run-time uncertainties from Section 2.1, except for the effectors, which are out of the scope of this dissertation.

With the overall contributions in Chapters 6 and 7, we showed the applicability of the theory of the SL in practice. However, with the evaluation as part of this paper, we have also shown that none of the original SL fusion operators support long-term knowledge aggregation. In response, we have proposed a new SL fusing operator, which is another

contribution as part of this paper.

Limitations: In this paper, although we evaluated the feasibility of different SL aggregation schemes, the effectiveness of our newly proposed aggregation scheme and the sensitivity of the proposed approach, we did not evaluate system adaptation according to the Definition 1 and the proposed Quality Function from Chapter 5.

Author Contribution: A. Petrovska identified the limitations of the preliminary results (see Chapter 6), precisely the limitations of the previous implementation of the system, based on which she identified the problem statement and the overall objective of this paper. A. Petrovska designed the initial draft of the solution, which M. Neuss implemented. A. Petrovska designed the experiments and M. Neuss conducted the experiments, and both authors jointly analysed the obtained results. The paper was written by A. Petrovska. I. Gerostathopoulos and A. Pretschner provided review and editing on the final drafts of the paper.

Run-time Reasoning from Uncertain Observations with Subjective Logic in Multi-Agent Self-Adaptive Cyber-Physical Systems

Ana Petrovska, Malte Neuss
Technical University of Munich
Munich, Germany
petrovsk@in.tum.de, neuss@in.tum.de

Ilias Gerostathopoulos
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
i.g.gerostathopoulos@vu.nl

Alexander Pretschner
Technical University of Munich
Munich, Germany
pretschn@in.tum.de

***Abstract*—Modern society has become increasingly reliant on the omnipresent cyber-physical systems (CPSs), making it paramount that the contemporary autonomous and decentralized CPSs (e. g., robots, drones and self-driving cars) act reliably despite their exposure to a variety of run-time uncertainties. The sources of uncertainties could be internal, i. e., originating from the systems themselves, or external—unpredictable environments. Self-adaptive CPSs (SACPSs) modify their behavior or structure at run-time in response to the uncertainties mentioned above. The adaptation relies on gained knowledge from the observations that the SACPSs make during their operation. As a result, to build the knowledge, the need for run-time observations aggregation and reasoning emerges since the observations made by decentralized CPSs are uncertain, partial, and potentially conflicting. In response, in this paper, we propose a novel methodological approach for deriving or aggregating knowledge from uncertain observations in SACPSs utilizing the Subjective Logic. The effectiveness of the proposed approach is demonstrated through extensive evaluation on an in-house, multi-agent system from the robotics domain.**

***Index Terms*—uncertainties, subjective logic, knowledge aggregation, self-adaptive systems, cyber-physical systems**

I. Introduction

Cyber-physical systems (CPSs) are software-intensive systems that control, communicate, and coordinate various processes in the physical and the digital worlds. Over the past decades, the boundary between the digital and physical has become increasingly blurry, accompanied by the rising prominence of CPSs. This process was further accelerated by the rapidly decreasing costs of embedded systems, which ultimately led to the omnipresence of CPSs as seen today. Nowadays, CPSs come in all shapes and sizes with applications as diverse as self-driving cars, smart homes, and entire robotic fleets, including autonomous robots and UAVs [6, 18].

As modern society increasingly relies on CPSs, these systems must act reliably even when faced with different run-time uncertainties. For instance, internal system uncertainties, i. e., different software or hardware faults and failures, as well as external uncertainties, e. g., uncertain and changing environments or *execution contexts* [21, 24, 28] in which the systems operate. Further, due to the limited range of their sensors, CPSs can only observe their context partially, which introduces another uncertainty dimension in the systems. A common approach to deal with uncertain and changing conditions at run-time while preserving the system's performance is to make CPSs self-adaptive.

In architecture-based self-adaptation [8, 12, 17, 40], a self-adaptive system—including self-adaptive CPS (SACPS)—is comprised of a *managed element* and an *adaptation logic*. The managed element can be either a software system or a CPS. If the SACPS consists of multiple autonomous and decentralized CPSs (i. e., managed elements), we refer to it as a Multi-Agent SACPS (MA-SACPS). On the other hand, the adaptation logic gives the managed element the ability to self-adapt and is commonly implemented using the MAPE-K feedback loop [17, 31, 32]. MAPE-K relies on four separate modules that *M*onitor, *A*nalyse, *P*lan and *E*xecute adaptations based on a shared *K*nowledge element. Furthermore, every system operates in a context. The context is the relevant part of the environment that can be observed, but not controlled. The knowledge (*K*) should keep models of both (1) the context in which the CPSs are operating, and (2) the CPSs themselves. These models should be continuously updated at run-time to reflect the dynamic changes in both the context and the CPSs; therefore, they need to be models at run-time (models@RT).

Models@RT [3, 4, 10, 38] are a promising approach for managing complexity at run-time, also considered as adaptation mechanisms for realizing self-adaptive systems. In past efforts, Floch et al. [10] and Bennaceur et al. [3] have identified the need for reasoning mechanisms as part of the models@RT, based on which the models are updated; however, no concrete solutions have been proposed. Concretely, the authors recognize as open research challenges: 1) the need for creating run-time models, and updating them in response to changes in the system and system's context; and 2) the need for reasoning (i. e., information or knowledge aggregation) based on which the models are updated.

Problem. MA-SACPSs are exposed to a variety of run-time uncertainties resulting in inaccurate and partial observations, which potentially lead to conflicting observations made by different CPSs. This can impact the overall performance of the adaptive system. Consequently, there is a need for reasoning

by effectively aggregating the different observations before updating *K*. This can be seen as an uncertainty resolution strategy applied at run-time.

Gaps. Although knowledge representation, aggregation and reasoning are essential for building MA-SACPSs [41], there is a scarcity of approaches for modeling *K* that allow capturing uncertain and conflicting observations from multiple, decentralized CPSs, to effectively aggregate the observations and eventually update the *K*. Additionally, in prior works, knowledge modeling has been typically treated as a domain-specific task [11], leading to ad-hoc solutions to its aggregation.

Solution. In this paper, we present a methodological approach for knowledge aggregation and reasoning in MA-SACPSs that is domain-independent and can deal with reasoning on uncertain, partial, and conflicting observations. Concretely, our approach uses Subjective Logic (SL) [14, 15] to update the knowledge in the adaptation logic at run-time by aggregating observations of the context made by each CPSs in a MA-SACPSs.

Contribution. Building on our previous work in which we introduced the idea of knowledge aggregation via Subjective Logic [25], in this paper we detail and extensively evaluate our fully developed approach. Succinctly, we make the following contributions:

(i) We present our fully developed SL-based approach for knowledge aggregation and reasoning in MA-SACPSs.
(ii) We provide an open-source implementation of a in-house ROS-based multi-robot system, which acts as a testbed for different scenarios involving uncertain and conflicting observations in the robotics domain.
(iii) We evaluate the effectiveness and sensitivity of the proposed SL-based approach through extensive controlled experiments.

Organization. In Section II we describe the class of use cases from the CPSs domain to which our approach is applicable, and a use case instantiation used as a running example throughout the rest of the paper. Section III summarizes the necessary background, before giving an overview of the approach in Section IV. In Section V we briefly describe the implementation of the approach, followed by the evaluation in Section VI. Section VII discusses related work, before concluding the paper in Section VIII.

II. Use Case

A. Class of Use Cases

Our proposed solution applies to any use case where multiple CPSs need to collaborate and coordinate processes assuming decentralized, partial and uncertain monitoring and centralized analysis. For example, mobile CPSs that autonomously traverse an environment to discover and attain tasks (e. g., robots or drones), or stationary agents that might have overlapping ranges of sensing (e. g., radio antennas). The CPSs could operate in two-dimensional (e. g., robots, self-driving cars) or three-dimensional environments (e. g., drones, UAVs). The dimension of the environment consequently defines how the context is modeled.

The complex and heterogeneous nature of CPSs often requires the SACPS to consist of multiple MAPE-K loops [25, 27, 41]. The use of multiple, interconnected MAPE-K loops leads to numerous challenges, one of which is the coordination of the control loops [39]. One possible solution is the use of design patterns [41] for distributed self-adaptive systems. In our paper, we assume that the considered CPSs are capable of independently monitoring the context in which they operate and simultaneously execute actions, i. e., performing the M and E phases of the MAPE-K on a *local level*. The decentralized phases of the MAPE-K inside the managed elements (the CPSs) are controlled by a single, centralized instance of planning (P), analysis (A), and knowledge (K) in the adaptation logic. Concretely, the MAPE-K loops inside the adaptation logic are structured according to the Master-Slave pattern (see Figure 1), previously proposed by Weyns [41]. To motivate the need for reasoning or knowledge aggregation, the centralized MAPE-K loop needs to reason on the uncertain, partial and potentially conflicting observations made by the decentralized monitoring, which, once aggregated, become knowledge.

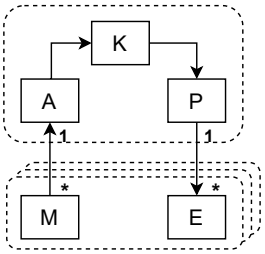


Fig. 1. Master-Slave pattern of the MAPE-K loop (updated from [41]).

B. Running example

The running example comprises of one or multiple robots operating in a room, in which dirt patches are continuously spawned with unknown location patterns and frequencies. The dirt patches represent cleaning tasks for the robots. Each robot is able to autonomously move to its destinations (i. e., the tasks' locations) while avoiding 1) static obstacles (e.g., walls, furniture, etc.), and 2) dynamic obstacles (e.g., other robots, humans) along its way. The robots fulfill their mission by discovering and cleaning the dirt i. e., completing the tasks without collision. They explore the room and detect new tasks in a distributed manner with a scanner, e. g., a LiDAR sensor. In addition to simply keeping the room clean, we also want to improve the performance or the quality of this cleaning process, e. g., cleaning the room in the shortest possible time.

However, the robots are subjected to external and internal uncertainties, manifested via the continuous appearance of tasks in the room—with unknown location patterns and different sensor uncertainties that cannot be anticipated during the design of the system. Due to the technical limitations of their sensors, the robots only monitor a limited range around them and can discover the newly appearing tasks only if they are within their range of observation. The partiality of the robots' observations introduces inefficiency to the overall system performance. Additionally, due to the sensor technology being imprecise and faulty, each robot might mistakenly sense a dirt task when there is none, and on the contrary, fail to sense an actual task. This potentially results in different robots holding

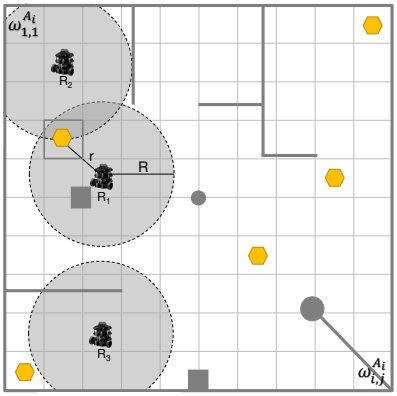


Fig. 2. Grid map and partial robots' observations, adopted from [25]

different opinions regarding the space they observe, which requires appropriate conflict resolution during the aggregation process.

The partial observations made by each robot are sent periodically to the centralized part of the adaptation logic, for instance, a `Cleaning Controller` (see Figure 3), in which the received observations are aggregated. The resulting aggregated knowledge, in terms of the presence of dirt tasks in the room, is the "best", most complete representation of the dynamic and uncertain context (i. e., the room) at a specific time. Optimizing the collective monitoring and analysis by obtaining the best representation of the state of the room, allows the performance of the full MAPE-K loop to also be improved and potentially optimized.

We model the context as a *grid map* with a size equal to the room's size (see Figure 2). A *context variable* ($w_x^{A_i}$) captures whether a dirt task occupies a specific cell of the grid map or not; therefore, there are as many context variables as cells in the map of the room. The figure also shows the robots' observation ranges and the tasks for the robots depicted in yellow hexagons.

III. Background

A. Run-time Uncertainties

In dynamic and adaptive systems, such as SACPSs, sources of uncertainty occur in one of the following three phases: requirements, design and run-time phase [30]. In this paper, we only consider uncertainty sources that occur in the run-time phase. We also classify the sources of run-time uncertainties as 1) *internal*—originating from the self-adaptive system itself, i. e., sensor failure, sensor imprecision, sensor noise and effectors, and 2) *external*, e. g., unpredictable environment.

In this work, we specifically focus on inaccuracy and inconsistency (stemming from sensor failure, imprecision, and sensor noise), and unpredictable environment. Ramirez et al. [30] define sensor failure as "a sensor inability to measure or report the value of a property", while inconsistency is defined as "two or more values of the same property that disagree with each other". Additionally, it is often infeasible 1) for the CPSs to observe the complete environment (i. e., context) in which they operate, due to the technical limitations of their sensors, e. g., limited sensor range, and 2) for the developers of the MA-SACPS to anticipate at system's design all the possible states of the context, which the systems will encounter during its operation. Consequently, the unpredictability of the environment will ultimately impart some partiality and uncertainty onto the MA-SACPS through its monitoring architecture.

B. Subjective Logic

When we assume an objective world, we can use binary logic to assert propositions about a state of the world to be either false or true. Nonetheless, it is practically impossible to determine with absolute certainty whether a given proposition is true or false. Through probability calculus, which takes argument probabilities in the range [0,1], we are able to reflect subjectivity by allowing propositions to be partially true. However, due to the lack of sufficient evidence, we are often unable to estimate probabilities with confidence. Furthermore, whenever the truth of a proposition is assessed, it is always done by an individual, and it cannot be considered to represent a general and objective belief. In order to reflect as faithfully as possible the perceived world in which we are immersed, a formalism to express degrees of uncertainty about beliefs is needed; said formalism shall also include belief ownership to reflect the subjective nature of beliefs [14, 15].

Subjective Logic (SL) [14, 15] is a framework for artificial reasoning, based on probabilistic logic and Dempster-Shafer theory of evidence [9, 33]. In recent years, SL has gained prominence because of its capability to deal with the degree of (un)certainty of propositions, inherently allowing 1) uncertainties representation as part of the fundamental building block of SL, called *Subjective Opinions* (see Section III-B1)), and 2) reasoning about the uncertainties through a process of *Belief Fusion* in which multiple Subjective Opinions are aggregated based on the selected fusion operator (see Section III-B2). For further explanation on SL please refer to [15, 25].

1) Subjective Opinions: A subjective opinion expresses a belief about a state variable X which takes its value from a domain $\mathbb{X}$. This domain represents all the possible states X can be. A binary domain $\mathbb{X} = \{x, \overline{x}\}$ is a type of domain that consists of two complementary states x and $\overline{x}$. In our running example, the state-space is a binary domain where the complementary states correspond to context variables (i. e., cells) in the grid map being occupied or unoccupied $\{x = \text{occupied}, \overline{x} = \text{unoccupied}\}$ by a task. If the domain of a subjective opinion is binomial, it is called a *binomial subjective opinion*. For the sake of brevity, the theoretical discussion of subjective logic is limited to binomial subjective opinions, as they are the only relevant in our case.

Definition 1 (Binomial Opinion [15])**.** Let $\mathbb{X} = \{x, \overline{x}\}$ be a binary domain with binomial random variable $X \in \mathbb{X}$. A binomial opinion about the truth of value X is the ordered quadruplet $w_x = (b_x, d_x, u_x, a_x)$, where the additivity requirement $b_x + d_x + u_x = 1$ is satisfied, and where the respective

parameters are defined as

b_x: belief mass in support of x being true (i.e. $X = x$),

d_x: disbelief mass in support of x being false (i.e. $X = \overline{x}$),

u_x: uncertainty mass representing the vacuity of evidence,

a_x: base rate, i. e., probability of x being true without any evidence.

Binomial opinions that have $u_x = 1$ and $u_x = 0$ are referred to as a vacuous and dogmatic opinions, respectively. The projected probability of a binomial opinion about value x is defined by: $P(x) = b_x + u_x a_x$.

2) Belief Fusion: Through a process of belief fusion, multiple opinions regarding the same proposition are merged or aggregated into a single, collective opinion. For instance, in our running example, multiple robots $R_1, R_2, \ldots, R_N$ issue opinions $w_x^{R_1}, w_x^{R_2}, \ldots, w_x^{A_N}$ for the same cell x, and these opinions need to be conveniently aggregated in a single opinion. Belief fusion can be realized using different operators [15, 37]: *averaging belief fusion*, *cumulative belief fusion*, *weighted belief fusion*, *consensus & compromise fusion*, and *belief constraint fusion operator*. Each of these fusion operators emphasizes different aspects when fusing multiple opinions. Subsequently, the choice of the operator depends on the aggregation's objective. In our use-case, the observations are made independently by multiple agents; thus, their opinions can be treated as independent pieces of evidence. Furthermore, compromises between said opinions are desired, such that the aggregated opinion is as accurate as possible. As a result, we choose the Cumulative Belief Fusion (CBF) and Consensus & Compromise Fusion (CCF) operators, although the implementation of the approach (further explained in Section V) supports all the other operators. In the following, we briefly summarize the two selected fusing operators.

Cumulative Belief Fusion (CBF) treats the individual opinions that are aggregated as independent pieces of evidence for the same proposition. This cumulatively increases the belief and/or disbelief value of the aggregated opinion while reducing its uncertainty. It is most suitable for combining multiple non-conflicting opinions. Namely, applying CBF to non-conflicting, uncertain opinions reduces the uncertainty of the resulting opinion.

Consensus & Compromise Fusion (CCF) maintains the shared belief masses between all the aggregated opinions. For conflicting opinions, a compromise is found, which has increased uncertainty. This operator is helpful for identifying a set of shared beliefs among all agents. The fused opinion would represent the set of causes that all opinions agree on.

The belief fusion aims to solve potential conflicting observations while increasing the observations' confidence according to the chosen fusion operator. To demonstrate this point, we provide a sample calculation for a pair of agreeing and disagreeing opinions w_x^1, w_x^2 on a same context variable in Table I.[1] We can see that when the opinions that are being aggregated agree, CBF creates an aggregated opinion whose belief increases that of the sources. CCF tries to find compromises between opinions, which is why their belief and disbelief values are in between the belief masses of ω_x^1 and ω_x^2. Additionally, with both fusion operators, we can observe improvement in the confidence of the aggregated observations, i. e., the uncertainty mass in the aggregated opinion decreases compared to the individual uncertainty masses of each robot's opinions. When the opinions are in conflict, CBF tries to maximize their belief masses. It is important to note that this process of aggregating conflicting opinions reduces the uncertainty of the aggregated opinion when using CBF, which can be undesirable. In contrast, CCF actively increases the uncertainty when opinions are conflicting. As a result, the resulting projected probability $P(x)$ is reduced when using CCF compared to CBF. This example showcases the different ways the two operators handle conflicts: whereas CBF essentially follows the strong opinion, CCF strives for a compromise. As we will see in our robotic use case, this difference manifests in the tradeoff between taking decisions fast (about whether a dirt is detected and should be cleaned) and taking decisions that are sound (since more than one robot agrees on the matter).

[1]For the mathematical definition of CBF and CCF see [16, 37].

TABLE I
AGGREGATING TWO OPINIONS w_x^1, w_x^2.

	Agreeing Opinions				Conflicting Opinions			
Param.	w_x^1	w_x^2	CBF	CCF	w_x^1	w_x^2	CBF	CCF
b_x	0.85	0.52	0.84	0.80	0.85	0.18	0.72	0.35
d_x	0.10	0.18	0.11	0.11	0.10	0.52	0.24	0.14
u_x	0.05	0.30	0.05	0.09	0.05	0.30	0.04	0.51
a_x	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50
$P(x)$	0.88	0.28	0.86	0.84	0.88	0.28	0.74	0.61

IV. APPROACH

In our approach, the MA-SACPS consists of several MAPE-K loops structured according to the master-slave pattern [41] As shown in Figure 3, the Monitor and Executor elements of the MAPE-K loops are distributed among the decentralized agents, controlled by centralized Analyzer, Planner and Knowledge elements, each element explained in the following.

A. Monitor

Each managed element (i. e., CPS) in the MA-SACPS comprises a Monitor component. The Monitor observes the context in which the CPS operates, and additionally, it does (a minimal) monitoring of the CPS itself, e. g., it reports on the current position of the system in the context. Since the CPS cannot observe the entire context, it only provides partial observations that are subjected to uncertainties. Based on these observations, the Subjective Opinion Creator—contained within the Monitor—creates *binomial subjective opinions* (see Section III-B1) about the context variables that are within the agent's range. These binomial subjective opinions are then independently forwarded to the centralized Analyzer.

In our running example each robot contains a Monitor component. It periodically senses the position of the robot, and detects the presence of dirt tasks in the room. As explained

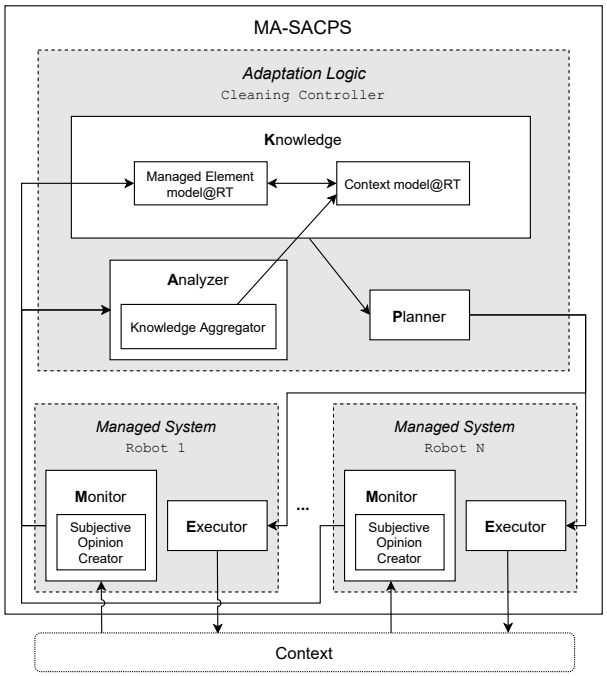


Fig. 3. High-level overview of the approach for reasoning in MA-SACPSs, adopted from [25].

above, each Monitor creates binomial subjective opinions about the context variables (is a cell occupied or not), for all the cells that are within the observation range of the robot at a specific point in time. The process of creating the subjective opinions is described in the following.

1) Subjective Opinion Creator: Every agent (A_i) creates a subjective opinion $w_x^{A_i}$ for every context variable x (i. e., grid cell) that is within its vision R. x can be either occupied or unoccupied by a task, and according to Definition 1 we define an opinion $w_x = (b_x, d_x, u_x, a_x)$ as following:

$$b_x = \begin{cases} 1 - u_X, \text{ for } x = occupied \\ 0.0, otherwise \end{cases} \qquad u_x = min\left(\frac{r}{R}, 0.99\right)$$

$$d_x = \begin{cases} 1 - u_X, \text{ for } x = unoccupied \\ 0.0, otherwise \end{cases} \qquad a_x = 0.5$$

where r is the distance between the agent and the cell and R is the agent's sensor range. This definition facilitates an uncertainty that increases linearly with the measurement distance up to an maximum value of 0.99. This value was chosen deliberately, since the observations at the edge of sensor range can be considered highly uncertain, but can still contribute towards the knowledge aggregation; hence, we assign an uncertainty value of 0.99 instead of a totally uncertain opinion (i. e., u_x = 1). Moreover, the polar nature between belief b_x and disbelief d_x was chosen to reflect the fact that agent can either detect a grid cell to be occupied or unoccupied by a task, and not a combination of both. The base rate a_x is always considered to be the default base rate for a binary domain, i. e., $a_x = 0.5$. Finally, no subjective opinions are issued for (i) cells which are occupied by static obstacles (e. g., walls), (ii) cells that lie outside of the detection range of the agent.

B. Analyzer

The Analyzer is a centralized component in which the reasoning takes place, with aim to solve any potential conflicting observations, while increasing the confidence of the observations according to the chosen subjective logic operator. The Analyzer contains the Knowledge Aggregator, which collects and fuses all the subjective opinions that are issued by the Subjective Opinion Creators in the decentralized agents. The aggregated opinions are used to update the run-time context model in K.

In our running example the Analyzer is a component in the `Cleaning Controller`. It combines the partial observations by aggregating the multiple binomial subjective opinions from different robot about the context variables that are within their sensing range. In particular, it is responsible for combining the opinions made by all the robots for all the cells that they are observing using the subjective logic operators. The process of aggregating multiple binomial subjective opinions is discussed next.

1) Knowledge Aggregator: This is a centralized node that aggregates the observations of the individual agents $(A_1, A_2, \dots A_N)$. The aggregated observations from all the agents are stored in the context model in the Knowledge, modelled as a grid map (see Section II-B). Upon system initialization, every context variable corresponding to each cell in the grid map is initialized with a vacuous subjective opinion: $w_x = \left(0, 0, 1, \frac{1}{2}\right)$. This initialization has the following two purposes: (i) a vacuous opinion will inform the analyzer that there is no previous knowledge about the context variables, and (ii) when the first knowledge aggregation is executed, the existing vacuous opinions do not influence the aggregation result.

The knowledge aggregation takes place every time the Subjective Opinion Creator of an agent publishes new subjective opinions for the context variables of the cells observed by the agent. Namely, the Knowledge Aggregator extracts the opinions that were previously stored for each context variable in the grid (w_x^g), and fuses them with the newly created opinion from each agent $(w_x^{A_i})$, resulting in a new aggregated opinion (w_x^{agg}) per context variable. The context variable x in the grid map is then updated based on aggregated opinion (w_x^{agg}), and this depicts is the overall process for updating the context model@RT (i. e., the grid map) in K. The process is executed with the same frequency for all the agents, and if multiple agents issue new opinions for the same context variable $(w_x^{A_1}, w_x^{A_2}, w_x^{A_3}, \dots)$ at the same time, then all of those opinion are together fused with the opinion from the grid (w_x^g).

The opinions are combined according to the following three schemes: CBF Scheme, CCF Scheme and Combination Scheme. The first two schemes are based on the CBF and CCF fusing operators, previously explained in Section III-B2:

$$\text{CBF Scheme}: \; w_x^{agg} = \text{CBF}(w_x^g, w_x^{A_i})$$
$$\text{CCF Scheme}: \; w_x^{agg} = \text{CCF}(w_x^g, w_x^{A_i})$$

However, preliminary testing revealed that these two SL operators cannot support long-term knowledge aggregation (see Section VI-B). In response, as part of this work, we have proposed a third scheme—Combination (Comb.) Scheme, defined as following:

$$w_x^{agg} = \begin{cases} \text{CCF}(w_x^g, w_x^{A_i}), & \text{if } u(w_x^g) < k \cap OT(w_x^g) \neq OT(w_x^{A_i}) \\ \text{CBF}(w_x^g, w_x^{A_i}), & \text{otherwise.} \end{cases}$$

where $u(w_x^g)$ is the uncertainty of context variable, k is a constant and $OT(w_x^{A_i})$ is the opinion type of the new opinion, defined as:

$$OT(w_x^{A_i}) = \begin{cases} \text{occupied}, & \text{if } b(w_x^{A_i}) \geq d(w_x^{A_i}) \\ \text{unoccupied}, & \text{if } b(w_x^{A_i}) < d(w_x^{A_i}), \end{cases}$$

where $b(w_x^{A_i})$ and $d(w_x^{A_i})$ are the belief and disbelief of the new opinion. In a nutshell, the combination scheme will always use the CBF operator, except when the new and grid opinions are conflicting and the uncertainty of the grid opinion is less than a given constant $k = 0.1$. We derived k numerically based on how many opinions need to be aggregated to change the opinion from unoccupied to occupied while still respecting the real-time capabilities of the knowledge aggregation (KA). Due to space limitations, the numerical calculation is not part of this paper.

C. Planner and Executor

The Planner is a centralized component in our approach that is responsible for selection of the adaptation actions or plans, which are later executed by the Executor components housed in every agent. The Planner relies on the aggregated knowledge, i. e., it uses the (updated) context model@RT to determine the actions for all the agents, in order for the MA-SACPS to adapt and accomplish the adaptation goals in the most efficient way. We do not prescribe how to perform the planning: any planning approach (e.g., rule-based, goal-oriented) can be used within our approach. Finally, each decentralized agent has an Executor component responsible for executing the actions previously determined by the Planner.

In our running example the Planner is part of the centralized `Cleaning Controller`, which receives as input the aggregated knowledge of the context. The aggregated knowledge, in terms of the presence of dirt tasks in the room, is the "best", most complete representation of the dynamic and uncertain context (i. e., the room) that is only partially observed at a specific time. Accordingly to it, the Planner assigns the discovered tasks to the robots. The outcome of the Planner is finally sent to the Executor in each robot. Each Executor contains an adaptive priority queue of the locations of the dirt tasks assigned to the specific robot. In our implementation, the queues of the robots are modified at run-time, based on the distance of the robot closest to the newly appeared task. Each robot picks the next task in its queue, and navigates autonomously to the corresponding cell.

V. Implementation

As part of this work, to investigate the usefulness of knowledge aggregation (KA) in MA-SACPSs, and to assess the correctness and the effectiveness of our proposed solution, we have implemented a testbed motivated by the running example from Section II-B. Since robotics is an intrinsically heterogeneous domain, setting up a *multi-robot* system is a challenge in itself. In response, in this paper, we provide a ROS-based, multi-robot system simulated in Gazebo [1, 19]. Namely, our implementation enables simulation of 1) a custom number of robots, and 2) the context in which they operate. Our running example concretely includes simulating a room with static (e. g., walls, corridors, furniture, etc.) and dynamic object i. e., the tasks that continuously appear for the robots in the room. The robots that we simulate are *TurtleBot 3 Burger*[2]. Furthermore, Gazebo relies on well-established physics engines, which enable simulations with high fidelity that closely resemble real-world robotics systems and their environments, with a physically correct representation of the robots, including their size and volume, frictions, as well as their sensors and actuators. Finally, the robots use Adaptive Monte Carlo Localization (AMCL) for localization and navigation.

Although we built a simulated robotics system, implementing realistic sensing (just how a real robot would sense its surroundings) was a prerequisite to constructing the subjective logic observations aggregation. As a result, the realistic sensing required more complex implementation to add different types of sensor uncertainties compared with, for example, "mocked" sensing in which modelling and adding the sensor noise would have been more simplistic. In the initial implementation of the robotic system, previously presented in [25], the sensor noise was sampled from a Gaussian distribution and only affected the *border* of the sensor beam. Although the former implementation did introduce some sensor noise in the system, it did not support the uncertainties from our running example (see Section II-B and Section III-A). Concretely, it was insufficient to support and prove the need for knowledge aggregation since it was incapable of creating conflicting observations within the sensor range. Consequently, we implemented a more sophisticated sensing model, in which the conflicting observations result from false positive (FP) tasks (tasks being observed by a robot that do not exist in reality) or false negative (FN) tasks (tasks not being observed when they in reality exist). Tasks that exist and the robots can observe are true positives (TP) in this setting.

The source code of the complete implementation and the installation instructions are available on the following link: https://github.com/tum-i4/Aggregatio. For the SL-based calculations, we used an open-source Java implementation[3].

[2]https://www.turtlebot.com/

[3]https://github.com/vs-uulm/subjective-logic-java

VI. Evaluation

In this section, we first explain the evaluation setup before discussing the obtained results.

We identified the following three research questions that guided our evaluation:

RQ1. How do the different SL aggregation schemes influence the KA in MA-SACPSs?

RQ2. Can SL-based KA in MA-SACPSs correct faulty measurements?

RQ3. How does the value of the threshold impact the SL-based KA in MA-SACPSs?

In RQ1, we investigate the *feasibility* of different SL aggregation schemes for KA in our approach. In RQ2, we evaluate the *effectiveness* of our approach's best aggregation scheme (derived from RQ1) on the core problem, i. e., reducing uncertainties and resolving conflicts introduced by inconsistent and faulty measurements. In RQ3, we investigate the *sensitivity* of our approach to the main parameter for decision making, i.e. the expected probability $P(x)$ of the context variables.

A. Experimental Setup

To answer the identified research questions, we have created three main experiments (see Table II), conducted based on 240 different simulation runs. Each experiment addressed one research question, and all the simulation runs lasted for 45 minutes in real-time. In all experiments, the dimension and the layout (e. g., corridors, walls, etc.) of the room in which the robots operate, along with the robots' initial positions and their sensor range, are held constant. The dimensions of the room and the sensor range of the robots were chosen in a way that at any given time the robots can only partially observe the room there are in. In particular, we chose a room size of $10 \times 10m$, since the robots can only detect obstacles up to a distance of $3.5m$.

Furthermore, we ran simulations under different settings, controlling: (i) the SL aggregation schemes for KA (part of the adaptation logic), (ii) the number and the properties of the robots (i. e., the managed elements), and (iii) the appearance of the tasks in the room (i. e., the context). With respect to the first point, we experimented with different *subjective logic operators* and *thresholds* values. The threshold is the projected probability $P(x)$ value for each detected task, necessary to be reached in order for the task to be promoted to a goal for the robots. With respect to the second point, we varied the *number of robots* and their sensing capabilities controlled by the *false positive (FP)* probability of each robot. The FP probability captures the percentage of faulty observations per robot. Finally, with respect to the environment, we varied the *rate of appearance of true positives (TP)* (in seconds) and the *location of appearance* of the tasks. A seed value controls the location of the appearance of the tasks, used to replicate the same distribution of tasks within the room in different experiments. To increase the validity of the results, we have run each experiment five times with different seed values. Using these variables, we explored the capabilities of SL-based KA and its impact on the performance of the MA-SACPS.

TABLE II
Design of the three experiments of the evaluation.

Parameter		Topic of Investigation RQ1	RQ2	RQ3
Experiment		1	2	3
KA		CBF, CCF, Comb.	[No, Comb., Comb.]	Comb.
Threshold		0.8	0.8	0.2, 0.4, 0.6, 0.8
No. of Robots		2	[1, 2, 5]	2
FP Prob.	R_1	0	0.1, 0.2, ... 0.9	[0.2, 0.5, 0.8]
	R_2	0	0.1	[0.2, 0.5, 0.8]
Rate of TP (s)		15, 60	60	60
Location (seed)		71, 72, ... 75	71, 72, ... 75	71, 72 ... 75

The rows in Table II represent the values of the variables used for a specific experiment. When multiple values are given per variable, then every possible combination is tested individually. For example, in Experiment 1 the CBF, CCF and Comb. operators are individually tested for all seed values. On the contrary, the square brackets indicate that these parameters are varied simultaneously. For example, in Experiment 2, when one robot is used, no knowledge aggregation is performed, whereas for two and five robots, the Comb. operator is used. Similarly, in Experiment 3 both robots R_1 and R_2 will have a FP probability of 0.2 in the first experiment, 0.5 in the next, etc. When using more then two robots, the additional robots will have the same FP probabilities as robot two (R_2). Finally, when Knowledge Aggregation is 'No', a task is classified as a goal after a single measurement, i. e., after it has been initially observed by one of the robots.

B. Experiment 1: Feasibility of different SL aggregation schemes

We initially aimed at the evaluation to use either CBF or CCF as operators for KA. Since CCF is desired when opinions are conflicting (see Section III-B2), the preliminary experiments quickly showed that it is hard to achieve extreme beliefs or disbeliefs when opinions are agreeing. This made it very difficult to surpass the required threshold needed for a task to be propagated as a goal and resulted in only a few tasks being completed. Subsequently, the focus shifted towards the CBF operator, which performed quite well. The cumulative aggregation method of CBF is ideal when opinions agree, and it is even capable of handling conflicting opinions to some degree. However, further analysis revealed that the task completion rate using CBF slowly deteriorates and eventually stops when running long simulations. We first explain this phenomenon via an analytical discussion and then show the results of the experiment.

1) Analytical Discussion: As discussed in Section III-B2, when fusing two opinions using CBF, the resulting opinion will have an uncertainty that is lower than that of the source opinions independent of whether the opinions agree or disagree. This property means that consecutive applications of CBF to fuse opinions issued by the robots will continuously decrease the uncertainty of the opinions of the context vari-

ables stored in the grid map. As the uncertainty decreases, the impact of a new opinion on the opinion in the grid map decreases as well, and after some time the grid opinions become too entrenched to change regardless of measurements. This demonstrates that this property is crucial when designing MA-SACPSs that need a support of a long-term KA.

The process that was described above can be demonstrated analytically with a few simple steps. First, consider three opinions: $w_x^{vac} = (0, 0, 1, 0.5)$, $w_x^{oc} = (0.7, 0, 0.3, 0.5)$, and $w_x^{unoc} = (0, 0.7, 0.3, 0.5)$ where the occupied and unoccupied opinions are in conflict with one another, and w_x^{agg} is the opinion of a context variable stored in the grid map. Initially, upon system initialization, $w_x^{agg} = w_x^{vac}$. Subsequently, the occupied opinion is aggregated with the base opinion, resulting in a new base opinion. This process is repeated 30 times. Afterward, the unoccupied opinion is aggregated with the base opinion for 30 times. These two steps of aggregating 30 occupied and 30 unoccupied opinions are repeated four times. Figure 4 plots the projected probability $P(x)$ of the w_x^{agg} opinion after each single aggregation (blue line). It can be seen from the figure that as the number of aggregated opinions increases, the effect of aggregating 30 occupied opinions diminishes significantly. The second peak occurs only 60 aggregations after the first peak and yet has a projected probability of about 40% less than the first peak after 30 aggregations. In our experiments, we consider a threshold of 80%, and opinions are issued every second. Thus, if a task is spawned at the first simulation minute (after 60 aggregations took place), even if the grid opinion has a projected probability of 50% and the robot measures the task for 30 seconds, it would still not reach the threshold and become a goal. As the figure shows, this gets even worse with time.

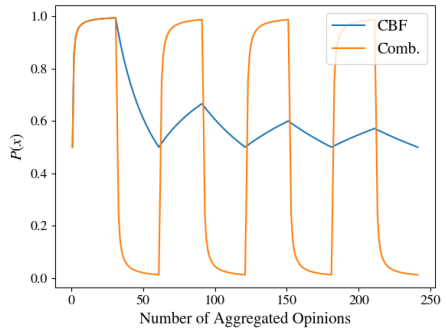


Fig. 4. The projected probability $P(x)$ shows that the effect of an individual opinion diminishes with time when using the CBF operator, and how the Comb. scheme overcomes this problem.

To address this issue, we have derived a new aggregation scheme (Comb.) using a combination of the CBF and CCF operators, which maintains the CBF operator's cumulative property but facilitates a faster switch in opinion when faced with contradicting opinions, and does not deteriorate with time. The proposed aggregation scheme was previously discussed in Section IV. Figure 4 shows that with the Comb.

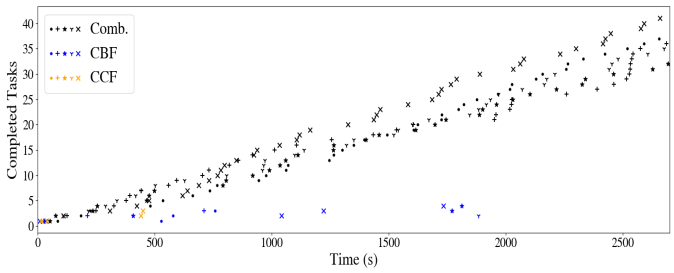


Fig. 5. Number of tasks completed using CBF, CCF and Comb. Schemes. The different symbols correspond to the different seed values.

scheme, the occupied and unoccupied opinions have equal impact on the opinion in the grid map and that their impact does not diminish over time.

2) Experimental Results: In this section, we experimentally assess RQ1. For that purpose, we set the FP probabilities to zero to demonstrate that the observed behaviors are solely related to the length of the simulation time and not to any uncertainty sources. The CBF, CCF, and Comb. aggregation schemes were each tested using the same five seeds with a threshold of 80% and a TP spawn interval of 15 and 60 seconds. The time at which the tasks have been completed was measured—Figure 5 shows the cumulative count of completed tasks over time. The different symbols correspond to the different seeds that have been tested at the spawning interval of 60 seconds. The results clearly show that independent of the location of the tasks, we can observe the same behavior: Comb. completes tasks at a relatively steady rate throughout the simulation. In contrast, the CBF operator only works well for the first few minutes before completely stopping. Lastly, the CCF operator performs the worst, which is expected as it is designed to find compromises and not aggregate observations cumulatively. The same behavior has been observed with a spawning interval of 15 seconds (plot not shown).

In summary, while addressing RQ1, we came up with the interesting finding that long-term KA is not feasible by using the original SL operators in isolation, based on which we proposed the new scheme—Comb. These experiments enabled us to prove some of the theoretical limitations of the CBF and CCF operators (Section III-B2) in an application from practice.

C. Experiment 2: Effectiveness of the KA

In this experiment, we investigate the effectiveness of the proposed combination scheme for KA by evaluating if the approach can correct the behavior of a robot with a faulty sensor. In this experiment, R_1 is considered to be the faulty robot subjected to a FP probability that ranges from 10% to 90% with a 10% step. Our baseline case is a single robot (R_1) that does not use KA. The other two cases consist of two and five robots and use the Comb. aggregation scheme for KA. All robots except R_1 are assumed to operate nominally with a FP probability of 10%. The measured metric is the number of FP and TP tasks that are completed by R_1.[4]

[4]By "completing" a FP task, we mean that the robot actually navigated to the place where the dirt was supposed to be.

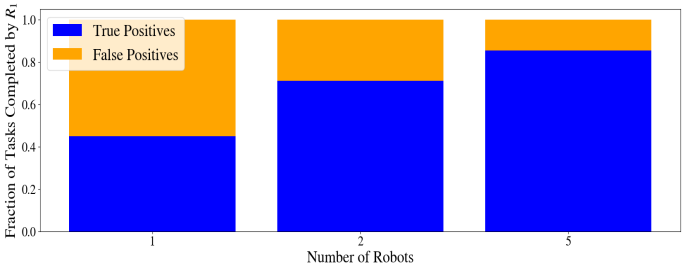


Fig. 6. The TP fraction of the tasks completed by robot 1 for a FP probability of 50%. The one robot case does not use KA whereas the combination aggregation scheme is used for two and five robot case. The results have been averaged over the five seeded runs.

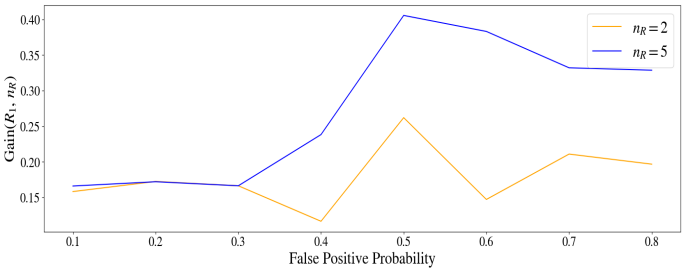


Fig. 7. The variation of the gain of robot 1 with FP probability when using two or five robots. The gain increases with FP probability, which demonstrates the effectiveness of the KA in correcting faulty observations.

Lets us first consider the tests when the FP probability of R_1 is 50%. Figure 6 plots the fraction of tasks completed by R_1 for the three different cases. As one would expect, about half of the completed tasks are TPs without KA, when only one robot is present. This fraction is significantly increased to about 70% when adding an additional robot and using KA. Adding even more robots further raises the TP fraction of tasks completed by R_1. This trend is expected since more robots make more independent observations, making it easier to correct faulty measurements. The bar charts further indicate that the gain in TP fraction per added robot is diminishing as the number of robots increases, which also makes sense as the extra vision gained per robot also decreases.

Instead of plotting a bar-chart for every FP probability case, we calculate the gain in TP fraction of robot R_i as follows:

$$\text{Gain}(R_i, n_R) = \frac{n_{TP}(R_i)}{n_{FP}(R_i)+n_{TP}(R_i)}\Big|_{n_R} - \frac{n_{TP}(R_i)}{n_{FP}(R_i)+n_{TP}(R_i)}\Big|_{1R}$$

where n_{TP} and n_{FP} is the number of TP and FP tasks completed by robot R_i and n_R is the number of robots in a multi-robot case. In particular, we calculate $\text{Gain}(R_1, n_R)$ for $n_R = 2$ and $n_R = 5$. The results in Figure 7 show that the gain in the TP fraction increases together with the FP probability for both cases—with two and five robots.

In summary, with respect to RQ2, we can conclude that SL-based KA enables the correction of faulty measurements made by a robot. In particular, even a single well-functioning robot can rectify the wrong measurements of another, faulty robot to a significantly extent. We have also observed that (i) increasing the number of well-functioning robots improves the accuracy of collective sensing; (ii) KA becomes increasingly effective as the faulty observations increase.

D. Experiment 3: Sensitivity analysis of the impact of the threshold value

The third experiment explores the impact of the threshold value on the behavior of the MA-SACPS. All simulations use two robots, and the FP probabilities are varied between 0.2, 0.5, and 0.8. Each of these variations is tested with four different thresholds ranging from 0.2 to 0.8 with a step of 0.2. As a metric, the number of completed TP and FP tasks are measured and averaged across the different seed values, based on which the average TP and FP fraction of completed tasks is calculated as follows:

$$\overline{TPF} = \frac{\overline{n}_{TP}}{\overline{n}_{FP} + \overline{n}_{TP}}, \quad \overline{FPF} = 1 - \overline{TPF}$$

where $\overline{n}_{TP}$ and $\overline{n}_{FP}$ are the average number of completed TP and FP tasks, and $N = n_{TP} + n_{FP}$ is the total number of tasks completed in a particular test. Moreover, the average number of completed tasks ($\overline{N}$) and the range in the number of completed tasks ($N_{max} - N_{min}$) are determined over five runs. The results from the experiment are depicted in Table III.

First, the table shows that increasing the threshold leads to an increased $\overline{TPF}$ for all the FP probabilities that have been tested. This indicates that the accuracy of the KA in MA-SACPS can be tuned using the threshold. This is expected, since the threshold dictates the minimum certainty the MA-SACPS must have to pursue the completion of a task. Nonetheless, the impact of the threshold on $\overline{TPF}$ is relatively small: changing it from 0.2 to 0.8 induces an increase of only 3.4% in $\overline{TPF}$ in the 0.2 probability case; 12.3% in the 0.5 probability case; and 3.8% in the 0.8 probability case. Furthermore, we can also observe another trend in the average number of completed tasks. Unsurprisingly, as the threshold increases, the number of completed tasks decreases.

In summary, with respect to RQ3, we conclude that the value of the threshold has an impact to SL-based KA, but a relative small one, and in any case a smaller impact than the number of collaborating robots. We also observed a clear trade-off between the accuracy of KA and the number of completed tasks.

E. Threads to validity

In order to have more realistic and comparable experiments, we kept the appearance rate of TP constant between runs since we wanted to explore how different FP probabilities and aggregation schemes respond to the same context (same ground truth of TP in the room). As a result, different FP probabilities result in a different number of tasks, i. e., higher FP probabilities generate more tasks in the room. This can be problematic for high FP probabilities as a high density of tasks restricts robots' observations, which severely limits the KA. These effects limit the validity of experiments at very high FP probabilities.

TABLE III
RESULTS FROM EXPERIMENT 3.

FP Prob.	Threshold	$\overline{TPF}$	$\overline{FPF}$	$\overline{N}$	$N_{max} - N_{min}$
0.2	0.2	0.769	0.231	53.6	7
	0.4	0.773	0.226	49.4	6
	0.6	0.794	0.206	45.6	5
	0.8	0.803	0.197	40.6	11
0.5	0.2	0.455	0.545	79.6	31
	0.4	0.485	0.515	41.8	64
	0.6	0.532	0.468	50.0	41
	0.8	0.578	0.422	25.6	57
0.8	0.2	0.205	0.795	93.6	68
	0.4	0.200	0.800	43.5	36
	0.6	0.219	0.781	47.4	86
	0.8	0.243	0.757	29.6	67

For example, in Experiment 2, the slight decreases at the end of Figure 7 are probably due to the aforementioned drop in the experiments' validity as the FP probability increases. In Experiment 3, we observed that the impact of the threshold is smaller than we have initially anticipated. Namely, at small FP probabilities (Table III), one would expect only a small improvement due to KA as faults rarely occur. On the contrary, at larger FP probabilities, one would expect a more considerable increase in the $\overline{TPF}$ than what is exhibited by the results. The same effect can be observed in the last column of Table III, which shows the difference between the maximum and the minimum number of tasks completed for the five different seed values. This clearly demonstrates that the results fluctuate a lot more as the FP probability increases. For example, for a threshold of 0.8 at a FP probability of 0.2 the range in N ($N_{max} - N_{min}$) is 27% of $\overline{N}$, whereas a FP probability of 0.8 with the same threshold has a range in N that is 226% of its $\overline{N}$. Subsequently, high FP probabilities require a lot more testing for accurate results. Furthermore, in Experiment 3, we concluded that there is a trade-off between the accuracy of KA and the number of tasks it completes. Based on this trade-off, one can conjecture that there is an optimal threshold that maximizes the number of completed TP tasks and varies with the FP and FN probabilities. In our experiment, the increase in $\overline{TPF}$ is too small to facilitate such a movement of the maximum; however, more extensive and statistically significant tests might prove this hypothesis.

VII. RELATED WORK

Knowledge/information aggregation. The need for knowledge aggregation arises in fields as varied as sensor fusion, expert system development and most prominently in multi-agent systems [26]. Across the field of multi-agent systems, knowledge is the information gained by agents' observations, often referred to as belief or belief base [26]. A belief is represented as propositional logic-based formalism [13, 26]. Grégoire and Konieczny in [13] present a survey about the approaches dealing with logic-based information fusion and discuss its relationship to multi-agent negotiation. Methods for belief merging are discussed in [20, 26, 34, 36]. These methods provide a general basis for knowledge aggregation; however, they have been only studied in the field of information systems. To the best of our knowledge, no knowledge aggregation technique has previously been used in the frame of SACPSs.

Sensor fusion. Munz and Dietmayer [22] proposes an approach to enhance the detection performance of a sensor fusion system measured in terms of detection rate versus false alarm rate. For that purpose, an algorithm is used which directly incorporates the DST-based sensory information. In [35], DST is also used to model the sources of uncertainty before applying the evidence fusion. However, the past approaches are mainly concerned with the aggregation of data in a single system, where the set of sensors are the multi-sources; rather than the aggregation of information or knowledge across multiple independent systems in a multi-agent system setup for self-adaptation purposes. Sensor fusion merges concrete sensor information, whereas, in our paper, we aggregate knowledge—which are two fundamentally different methods. Additionally, SL has not been used as a framework for fusing sensor information before, both generally and in the frame of self-adaptive systems.

Mitigating uncertainties in self-adaptive systems. As discussed in Section III-A, in this work, we focus on the following run-time uncertainties: sensor inconsistency, sensor failure, and unpredictable environment. Although the past literature proposes different mitigation strategies for sensor failure [5, 11] and unpredictable environment [2, 5, 7, 11, 23, 29], to the best of our knowledge, no solutions have previously tackled sensor inconsistency. Almost all existing uncertainties mitigation strategies are based upon one or multiple feedback loops, i. e., the different MAPE-K phases [5] that are not modified beyond their design-time specifications. Also, they are often designed for a specific application [23]. The most prominent framework for mitigating uncertainties is Rainbow [11], which focuses on architectural reusability. Rainbow focuses on isolating the feedback loop from the managed system as much as possible, enabling system-independent but knowledge-specific infrastructure. Furthermore, the framework does not support run-time modification of the adaptation logic, which disables the framework to deal with uncertain situations and changing conditions that were not anticipated during its design, as we do in our paper. In comparison to the past solutions, our approach presents a run-time uncertainly resolution strategy that tackles sensor inconsistency, as well as sensor failure and unpredictable environments.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a domain-independent methodological approach for knowledge aggregation and reasoning of decentralized monitoring in MA-SACPS, which inherently produce partial, faulty and potentially conflicting context observations. The proposed approach allows capturing uncertainty at run-time on a local level, and effective reasoning and knowledge aggregation for global decision-making. The conducted experiments revealed that i) no single SL operator

is capable of providing long-term real-time KA capabilities, ii) the proposed SL-based KA approach is capable of correcting the observations of a faulty agent, and iii) there is a trade-off between the number of tasks that are completed and the accuracy of the agents, and the threshold can be used to tune the accuracy of KA to a desired level.

As future work, we want to find a threshold, if one exists, that maximizes the number of completed TP tasks, which varies with the FP probabilities. Since the FP probabilities result from the agents' interaction with the environment, the existence of a maximum would render the threshold as another parameter that can be optimized in real-time, ultimately posing another possibility for self-adaptation. Alternatively, one can also evaluate different and more advanced combinations of SL operators and different room sizes and layouts to improve the performance of KA even further, or investigate how KA with SL differs from reasoning with other non-monotonic logics.

REFERENCES

[1] Carlos E. Aguero, Nate Koenig, Ian Chen, Hugo Boyer, Steven Peters, John Hsu, Brian Gerkey, Steffi Paepcke, Jose L. Rivero, Justin Manzo, Eric Krotkov, and Gill Pratt. Inside the virtual robotics challenge. 12:494–506, 2015. ISSN 1545-5955. doi: 10.1109/TASE.2014.2368997.

[2] Luciano Baresi, Liliana Pasquale, and Paola Spoletini. Fuzzy goals for requirements-driven adaptation. In *2010 18th IEEE International Requirements Engineering Conference*, pages 125–134. IEEE, 2010.

[3] Amel Bennaceur, Robert France, Giordano Tamburrelli, Thomas Vogel, Pieter J Mosterman, Walter Cazzola, Fabio M Costa, Alfonso Pierantonio, Matthias Tichy, Mehmet Akşit, et al. Mechanisms for leveraging models at runtime in self-adaptive software. In *Models@ run. time*, pages 19–46. Springer, 2014.

[4] Gordon Blair, Nelly Bencomo, and Robert B France. Models@ run. time. *Computer*, 42(10):22–27, 2009.

[5] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*, pages 48–70. Springer, 2009.

[6] Hon Chen. Applications of cyber-physical system: A literature review. *Journal of Industrial Integration and Management*, 2017. doi: 10.1142/S2424862217500129.

[7] Betty HC Cheng, Pete Sawyer, Nelly Bencomo, and Jon Whittle. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *International Conference on Model Driven Engineering Languages and Systems*, pages 468–483. Springer, 2009.

[8] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, pages 2–8, 2006.

[9] A. P. DEMPSTER. A Generalization of Bayesian Inference Author (s): A . P . Dempster Source : Journal of the Royal Statistical Society . Series B (Methodological), Vol . 30 , No . 2 Published by : Wiley for the Royal Statistical Society Stable URL : http://www.jstor.o. 30 (2):205–247, 1968.

[10] Jacqueline Floch, Svein Hallsteinsen, Erlend Stav, Frank Eliassen, Ketil Lund, and Eli Gjorven. Using architecture models for runtime adaptability. *IEEE software*, 23(2): 62–70, 2006.

[11] David Garlan, S-W Cheng, A-C Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[12] David Garlan, Bradley Schmerl, and Shang-Wen Cheng. Software architecture-based self-adaptation. In *Autonomic computing and networking*, pages 31–55. Springer, 2009.

[13] Eric Grégoire and Sébastien Konieczny. Logic-based approaches to information fusion. *Information Fusion*, 7(1):4–18, 2006. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2005.08.001. URL http://www.sciencedirect.com/science/article/pii/S1566253505000771.

[14] Audun Jøsang. A Logic for Uncertain Probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(3):271–311, 2001.

[15] Audun Jøsang. *Subjective Logic*. Springer International Publishing Switzerland, 2016. ISBN 978-3-319-42335-7. doi: 10.1007/978-3-319-42337-1. URL http://link.springer.com/10.1007/978-3-319-42337-1.

[16] Audun Jøsang, Dongxia Wang, and Jie Zhang. Multi-source fusion in subjective logic. *20th International Conference on Information Fusion, Fusion 2017 - Proceedings*, 2017. doi: 10.23919/ICIF.2017.8009820.

[17] Jeffrey O. Kephart and David M. Chess. The vision ofautonomic computing. *Computer 36*, pages 43–50, 2003. doi: https://doi.org/10.1046/j.1365-2745.2002.00730.x.

[18] Junsung Kim, Hyoseung Kim, Karthik Lakshmanan, and Ragunathan Rajkumar. Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car. *2013 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2013.

[19] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, pages 2149–2154. IEEE, 2004. ISBN 0-7803-8463-6. doi: 10.1109/IROS.2004.1389727.

[20] Sébastien Konieczny and Ramón Pino Pérez. Merging information under constraints: A logical framework. *Journal of Logic and Computation*, 12(5):773–808, 2002. ISSN 0955792X. doi: 10.1093/logcom/12.5.773.

[21] Sara Mahdavi-Hezavehi, Paris Avgeriou, and Danny

Weyns. A classification framework of uncertainty in architecture-based self-adaptive systems with multiple quality requirements. *Managing Trade-Offs in Adaptable Software Architectures*, pages 45–77, 2017.

[22] Michael Munz and Klaus Dietmayer. Using Dempster-Shafer-based modeling of object existence evidence in sensor fusion systems for advanced driver assistance systems. *IEEE Intelligent Vehicles Symposium, Proceedings*, (Iv):776–781, 2011. doi: 10.1109/IVS.2011.5940463.

[23] Peyman Oreizy, Michael M Gorlick, Richard N Taylor, Dennis Heimhigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S Rosenblum, and Alexander L Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and Their Applications*, 14(3):54–62, 1999.

[24] Ana Petrovska and Alexander Pretschner. Learning approach for smart self-adaptive cyber-physical systems. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 234–236. IEEE, 2019.

[25] Ana Petrovska, Sergio Quijano, Ilias Gerostathopoulos, and Alexander Pretschner. Knowledge aggregation with subjective logic in multi-agent self-adaptive cyber-physical systems. In Shinichi Honiden, Elisabetta Di Nitto, and Radu Calinescu, editors, *SEAMS '20: IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Seoul, Republic of Korea, 29 June - 3 July, 2020*, pages 149–155. ACM, 2020. doi: 10.1145/3387939.3391600. URL https://doi.org/10.1145/3387939.3391600.

[26] Gabriella Pigozzi and Stephan Hartmann. Aggregation in multiagent systems and the problem of truth-tracking. *Proceedings of the International Conference on Autonomous Agents*, (May 2014):219–221, 2007. doi: 10.1145/1329125.1329245.

[27] Mariachiara Puviani, Giacomo Cabri, and Franco Zambonelli. A taxonomy of architectural patterns for self-adaptive systems. In *Proceedings of the International C* Conference on Computer Science and Software Engineering*, pages 77–85. ACM, 2013.

[28] Federico Quin, Thomas Bamelis, Singh Buttar Sarpreet, and Sam Michiels. Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning. *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 1–12, 2019.

[29] Andres J Ramirez, Adam C Jensen, Betty HC Cheng, and David B Knoester. Automatically exploring how uncertainty impacts behavior of dynamically adaptive systems. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 568–571. IEEE, 2011.

[30] Andres J Ramirez, Adam C Jensen, and Betty HC Cheng. A taxonomy of uncertainty for dynamically adaptive systems. In *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 99–108. IEEE, 2012.

[31] De Lemos Rogerio, Holger Giese, Hausi A Mller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, and Thomas Vogel. Software engineering for self-adaptive systems: A second research roadmap. *Software Engineering for Self-Adaptive Systems III. Assurances*, pages 2–13, 2013.

[32] De Lemos Rogerio, David Garlan, Carlo Ghezzi, Holger Giese, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Danny Weyns, Luciano Baresi, and Nelly Bencomo. Software engineering for self-adaptive systems: Research challenges in the provision of assurances. *Software Engineering for Self-Adaptive Systems III. Assurances*, pages 3–30, 2017.

[33] Glenn Shafer. A mathematical theory of evidence. *Princeton University Press*, 1976.

[34] Luciano H. Tamargo, Alejandro J. García, Marcelo A. Falappa, and Guillermo R. Simari. Modeling knowledge dynamics in multi-agent systems based on informants. *Knowledge Engineering Review*, 27(1):87–114, 2012. ISSN 02698889. doi: 10.1017/S0269888912000021.

[35] Yongchuan Tang, Deyun Zhou, Zichang He, and Shuai Xu. An improved belief entropybased uncertainty management approach for sensor data fusion. *International Journal of Distributed Sensor Networks*, 13(7), 2017. ISSN 15501477. doi: 10.1177/1550147717718497.

[36] Trong Hieu Tran, Ngoc Thanh Nguyen, and Quoc Bao Vo. Axiomatic characterization of belief merging by negotiation. *Multimedia Tools and Applications*, 65 (1):133–159, 2013. ISSN 13807501. doi: 10.1007/s11042-012-1136-7.

[37] Rens W. Van Der Heijden, Henning Kopp, and Frank Kargl. Multi-Source Fusion Operations in Subjective Logic. *2018 21st International Conference on Information Fusion, FUSION 2018*, pages 1990–1997, 2018. doi: 10.23919/ICIF.2018.8455615.

[38] Thomas Vogel, Andreas Seibel, and Holger Giese. The role of models and megamodels at runtime. In *International Conference on Model Driven Engineering Languages and Systems*, pages 224–238. Springer, 2010.

[39] Danny Weyns. Software engineering of self-adaptive systems. In *Handbook of Software Engineering*, pages 399–443. Springer, 2019.

[40] Danny Weyns and Tanvir Ahmad. Claims and evidence for architecture-based self-adaptation: a systematic literature review. In *European Conference on Software Architecture*, pages 249–265. Springer, 2013.

[41] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M Göschka. On patterns for decentralized control in self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*, pages 76–107. Springer, 2013.

Part III

Related Work and Conclusion

8 Related Work

This chapter presents prior related efforts on defining self-adaptive systems, engineering self-adaptive systems and finally, mitigating uncertainties in self-adaptive system. Parts of this chapter have been published in peer-reviewed publications [75, 97, 98, 96, 94] co-authored by the author of this thesis.

8.1 Defining System Adaptation and Self-Adaptive Systems

In the following section, we examine the available informal and formal definitions of system adaptation and self-adaptive systems, followed by a short discussion on the relation of self-adaptive systems with autonomous, context-aware and self-aware systems.

8.1.1 Informal Definitions of Self-Adaptive Systems

The ideas of self-adaptive systems have been rapidly growing for the past two decades, and many attempts in the literature have been made to define this concept. As discussed in the previous chapters of this thesis, there is still no general agreement on a broadly accepted definition of self-adaptive systems in this research field. However, several prior works have made efforts to define self-adaptive systems, both informally and formally.

Table 8.1 summarizes some of the existing informal definitions.

Study	Definition
[77]	"Self-adaptive software is informed about its mission, construction and behaviour, and it evaluates its own behaviour and changes behaviour when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible."
[92]	"Self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation."
[117]	"The aim of self-adaptation is to let the system collect additional data about the uncertainties during operation. The system uses the additional data to resolve uncertainties, to reason about itself, and based on its goals to reconfigure or adjust itself to satisfy the changing conditions, or if necessary to degrade gracefully."
[32]	"A self-adaptive system is a system that is able to adjust its behaviour in response to their perception of the environment and the system itself."

[73]	"A self-adaptive system is able to automatically modify itself in response to changes in its operating environment. The modification is done by adjusting attributes (parameters) or artifacts of the system in response to changes in the system itself or in its environment."
[120]	"Self-adaptive software systems are an emerging class of systems that adjust their behavior at runtime to achieve certain functional or quality of service objectives."
[17]	"The core concept behind adaptability is the general ability to change a system's observable behavior, structure, or realization basically without users' interaction."
[22]	"A software system is called self-adaptive if it can modify its behaviour as a reaction to a change in its context of execution, understood in the widest possible way, including both the external environment and the internal state of the system itself."
[78]	Self-adapting is "a system's ability to modify itself (self-adjust) in reaction to changes in its execution context or external environment, in order to continue to meet its business objectives despite such changes."
[23]	"Self-adaptive systems have been introduced to manage situations where software systems operate under continuous perturbations due to the unpredicted behaviours of their clients and the occurrence of exogenous changes in the environment in which they operate."
[58]	"Self-adaptive systems are systems that adapt to dynamics by reconfiguring their structure and modifying their behaviour at runtime with little or no human intervention."
[54]	"Self-adaptive systems adapt their structure and behaviour to cope with changing environment conditions."

Table 8.1: *Informal definitions of self-adaptive systems in the literature.*

Although all of these definitions might look intuitive and clear at first glance, all of them (except the first definition by Laddaga) are affected by the limitations that were previously elaborated in more detail in Sections 1.1.3 and 1.3, where we used the informal definition from Broy as an analysis reference [17]. Concretely, none of the informal definition distinguishes when a system functions and when it adapts (i. e., the difference between non-adaptive and adaptive systems), what is the minimum criteria and the prerequisites for a system to be self-adaptive, nor in general provides a more specific semantic of system adaptation or self-adaptive systems.

Besides the informal definitions listed in Table 8.1, some other studies have taken a different approach towards defining self-adaptive systems. For example, Weyns in [116] states that although there is no general consensus on the definition of self-adaptation, there are two common interpretations, also called *principles*, that determine what self-adaptive systems are. The proposed principles, referred to as external and internal principle, are complementary to each other and explained as follows.

According to the *external principle*, "a self-adaptive system is a system that can handle

changes and uncertainties in its environment, the system itself, and its goals autonomously (i. e., without or with minimal required human intervention) [116]." According to the *internal principle*, "a self-adaptive system comprises two distinct parts: the first part interacts with the environment and is responsible for the domain concerns—i. e., the concerns of users for which the system is built; the second part consists of a feedback loop that interacts with the first part (and monitors its environment) and is responsible for the adaptation concerns—i. e., concerns about the domain concerns [116]."

The external principle considers the self-adaptive system as a black box, just as an external observer would examine the system. This principle puts two aspects in focus:

1. the changes and uncertainties in the system, its environment and the (system) goals,
2. and that in self-adaptation, the *self-* prefix indicates that the system performs the adaptation autonomously, without or with minimal human intervention.

In the external principle, it needs to be clarified from where the changes and uncertainties in the goals originate, since it is left unclear what the goals are and where they are defined. Specifically, the notion of goals in the description of the external principle remains under-specified, and a further explanation of the form that the goals take, who sets them, and how and when they are elicited remains open and ambiguous. Furthermore, the aspect of human and human intervention are as well left unspecific. For example, is the human the end-user, the developer or the maintainer of the system who needs to intervene with the system in case of system faults and failures in order for the system to stay operational? Finally, from the external principle, it is unclear what the author understands by the term handling, concretely, by self-adaptive systems handling the changes and the uncertainties.

On the contrary, the internal principle looks at the system from an engineering standpoint, e. g., how an engineer would build a self-adaptive system. If we compare the internal principle and the architectures of autonomous systems that we proposed in Chapter 3, we can observe the same design pattern for engineering self-adaptive and autonomous systems. Specifically, the system (also referred to as a managed element as part of a self-adaptive system) is enriched with a control logic (the adaptation logic as part of a self-adaptive system). Therefore, from an architectural point of view there is no difference between a self-adaptive system according to the second principle and an autonomous system according to our definitions in Chapter 3.

A potential essential aspect to differentiate autonomous from self-adaptive systems in the internal principle is the separation of concerns between the so-called first part (i. e., the managed element), from the so-called second part (i. e., the adaptation logic). Concretely, Weyns states that the managed element is responsible for the domain concerns, and the adaptation logic is responsible for the adaptation concerns; however, without even clarifying what he precisely understands under adaptation concerns, or in general, adaptation. Please note that the author in [116] uses the terms *system concerns* (in the internal principle) and *system goals* (in the external principle) interchangeably. In summary, both principles fall short of increasing the semantic clarity of self-adaptive systems.

Summary. The existing informal definitions of self-adaptive systems from the prior works remain underspecified since they primarily rely on the intuitive meaning of the language. Over the years, not even an informal working definition was established and

accepted in the community on a broader scale. As a result, in software and systems engineering, or science in general, a definition with more rigour is required to understand better *what* are self-adaptive systems and, respectively, the emerging engineering processes and implications thereof. This further supports the necessity for formalized definitions and a formalized world of concepts.

8.1.2 Analysis of the Formal Definitions of System Adaptation and Self-Adaptive Systems

To the best of our knowledge, only a few approaches in the literature provide a formalized world of concepts for system adaptation and self-adaptive systems. We summarize them in the following section.

One of the first efforts toward formally defining adaptive behaviour was done by Zhang and Cheng [129]. In their work, the authors propose a model-driven software development process for dynamically adaptive programs, focusing on behavioural modelling. Concretely, they introduce an approach to create formal models for the behaviour of adaptive programs. According to the authors, 1) in order for an adaptive program to be trusted, it is important to have a mechanism to ensure that the program functions correctly during and after adaptations, and 2) an adaptation can only be performed safely only when the program is in a quiescent state. This all results in adaptive programs being generally more difficult to specify, verify and validate due to their high complexity, especially in multi-threaded adaptations where the program behaviour results from the collaborative behaviour of multiple threads. Although the authors' goal is to define an adaptive program, they do not aim to answer what they understand under the notion of *adaptation* as part of their work. Also, it is important to point out that adaptation is not necessarily an emerging property from a collaboration, as the authors consider as part of their work, and it should be treated and defined as a separate concept. In their formal representation of adaptive programs, a *program* is represented by a state machine (a finite state automaton) that exhibits certain behaviour and operates in certain domains. A dynamically adaptive program operates in different domains and changes its behaviour at run-time in response to domain changes. According to the authors, an adaptive program is a program whose state space can be separated into several disjoint programs, where each of the programs exhibits different behaviour and operates in a different domain. The states and transitions connecting one region to another are adaptation sets.

As part of their work, the authors do not formalize system adaptation in general, but they illustrate the adaptation specification process for three types of "adaptive behaviour" in adaptive programs by modelling an audio streaming protocol with Petri nets. To formalize adaptive programs, the authors use prior works on specifying dynamic systems architectures [5, 15] as the foundation for their formalisation, often using the terms adaptive and dynamic interchangeably without providing a clear distinction between these terms. Although the presented motivation for the work: 1) the need for the program to function correctly during and after the adaptation and 2) the idea of a quiescent state, are indeed relevant for (self-)adaptive systems, it is not clear from their proposed specifications how adaptive programs differ from dynamic systems, and in general, how their specifications

support the characterization and the engineering of (self-)adaptive systems on a bigger scale. As emphasized as part of this dissertation, defining system adaptation can only be done within an appropriate framing, which considers aspects such as changes and uncertainties, adaptation triggers, context, domain and adaptation goals, and quality. Unfortunately, all of these aspects are entirely out of the scope in [129].

In two separate works, Broy et al. [20] and Bruni et al. [22] try to answer the question of how the adaptive systems differ from the "ordinary" systems, which are considered non-adaptive. Concretely, Broy et al. [20] aim at defining adaptive system behaviour while differentiating interaction patterns between the system, a subject (a user or other technical system that interacts with the system), and the environment (modelled as context). The authors claim that the adaptive behaviour of the system can be differentiated only by considering and observing the environment in which the system operates. In other words, it cannot be argued if something adapts in isolation from the environment (i. e., the context) and its state, which is a conclusion that we have also come to as part of our work and is one aspect of our framing, proposed in Chapter 5. Additionally, in [20], the authors classify the system inputs into direct/explicit inputs and indirect/implicit inputs. Namely, they assume that the system always receives the user inputs explicitly and that a user experiences an adaptive system behaviour if the system reaction resulting from the user input is additionally determined by some additional information about the environment received through the implicit inputs. Furthermore, the authors identify four types of observable system behaviour (i. e., adaptive behaviour) with respect to the user: non-adaptive, non-transparent adaptive, transparent adaptive and diverted adaptive behaviour. This work is based on the formalism of Focus [18], which the authors use to model the different types of adaptive behaviour through formalizing the interactions between the subject, the system and the environment. Although the authors identify the consideration of the context state and the system state as relevant for the adaptation, and they reach to some sound conclusions (e. g., the adaptation of the system can only be determined within the frame of context in which the system operates), the definitions in their work still lack completeness, i. e., they exclude all the other aspects that are necessary and essential for a holistic definition of system adaptation (see Chapter 4). Despite these limitations, the work by Broy et al. [20] still proposes a unique interpretation of adaptive system behaviour. Concretely, his paper differs from all the other works in the literature, and it is the first existing work that puts the definition of system adaptation in a different perspective, identifying that adaptation as a system property cannot be defined in isolation from other notions (in particular, the context as part of their paper).

Bruni et al. [22] propose a conceptual framework for adaptation, in which they assign a central role to control data, which governs the adaptive behaviour of a component. Namely, they define adaptation as a run-time modification of the control data. Consequently, the authors consider a component as self-adaptive if it can modify its own control data at run-time. Furthermore, the authors propose a simple Labelled Transition Systems (LTS) formal model based on their conceptual framework. They discuss adaptable vs non-adaptable components, self-adaptive components, and knowledge-based adaptation, in which they recognize the context as the observable part of the environment. To this point, the authors mention the notion of partiality but only very briefly. Although the

authors make a considerable effort towards tackling such a challenging issue as defining adaptation, the proposed formalization of their concepts and even the proposed conceptual framework are still very abstract to be helpful towards a design or a later implementation of a self-adaptive system. Concretely, it is left general and is not elaborated clearly on what the authors understand under the notion of control data, how one can identify control data in the system, how the system is influenced by the control data and the structure of the control data. Unfortunately, this level of abstraction and the fuzziness of the central notion of their formalisms, i. e., the control data, renders the complete formalization non-usable. Also, the authors do not discuss any methodological or architectural implications of their formalisms nor consider any of the essential aspects we elicit in this dissertation that are crucial for defining these systems.

Weyns et al. [120] and Arcaini et al. [8] propose formally specified models for designing self-adaptive software systems, concretely distributed/decentralized self-adaptive systems. The authors propose a FOrmal Reference Model for Self-adaptation (FORMS), which enables precise descriptions of the architectural characteristics of distributed self-adaptive software systems in the system's early design phases. The model is formally specified in the Z notation. Although FORMS had and continues to have a notable impact in the community, it does not define what system adaptation is, nor does it help differentiate between self-adaptive and non-adaptive systems. Furthermore, FORMS focuses on the formalization of the structural aspect of self-adaptive systems and does not provide a formalization of the behavioural semantics of the systems. Also, the authors do not consider the need and importance of considering and representing the context as part of the solution for engineering self-adaptive systems that they propose. Namely, the idea behind the dynamics of these systems—concretely the context and system changes—that could not be anticipated during the systems' design, and all the resulting implications, including the need for different mechanisms for knowledge representation, reasoning, etc., are completely excluded in FORMS.

Arcaini et al. [8] show how to model the adaptation logic and the behaviour of self-adaptive systems by utilizing the theoretical framework of the multi-agent Abstract State Machines (ASM). Concretely, the authors represent multiple decentralized MAPE-K loops formalized in terms of ASM transition rules. Although the authors aim at modelling self-adaptive systems, they do not explain what the adaptive behaviour of a system means for them. However, it is important to point out that this is the only related work where the authors mention and put the focus on the system's functionality in some way, implicitly implying and framing the adaptation as a property that modifies the system's nominal function. Namely, this paper distinguishes between the logic necessary for self-adaptation (in their case, the various decentralized MAPE-K loops) and the system's functional logic, which provides the system's functionality. Similarly as to some conclusions that we have reached as part of our work, the authors of [8] emphasize the importance of the knowledge (i. e., the knowledge component), which needs to be built according to some relevant aspects for the concrete adaptation. However, the authors do not elaborate on the knowledge beyond this explanation nor provide any solutions towards modelling and reasoning (i. e., updating the knowledge at run-time). Finally, the formalizations in [8] leverage the aspect of decentralization and consider adaptation as a result of the collaborative behaviour of

multiple systems (i. e., managing systems as part of MAPE-K). However, defining the system adaptation should be independent of the nature of the system. Our definitions are applicable to single and multiple systems with distributed or centralized nature. A thorough consideration of the quality and phenomena of the context in which the systems operate, as well as the consideration of the adaptation goals and their separation from the business goals, are crucial prerequisites for building systems with self-adaptive capabilities. However, none of the two papers by Weyns et al. [120] and Arcaini et al. [8] considers them in their solution.

Context has been more carefully considered in the works proposed by Abusair et al. [2] and Bucchiarone and Mongiello [23]. Abusair et al. [2] bring the system quality and the context changes, including the need for context awareness for system adaptation to the forefront, by focusing on how context changes may affect system quality. The authors argue that the systems may greatly benefit from context awareness to adapt to the context changes. This goes in the direction of some of the conclusions from this dissertation related to the minimal requirements for a system to be self-adaptive. In other words, the knowledge component in the adaptation logic that consists of models of the system (enabling self-awareness) and models of the context (enabling context awareness). Please note that when we write this we only refer to passive self-adaptation (as discussed in Appendix A)—the minimal set of requirements for the active self-adaptation differs. The approach proposed by Abusair et al. also leverages software quality under different contextual situations to determine the best adaptation in a given context by using a reward function. This could be considered as some quantification of the quality aspect in their paper, which is exacctly the idea behind the Quality Function in our contributions. However, the authors do not make any separation between the business and the adaptation goals, nor define what the authors understand under system adaptation. Namely, although the authors relate context awareness to adaptation, the paper does not provide any formal foundations or answers to the main problem that we tackle in this thesis, i. e., what is system adaptation and, respectively, self-adaptive systems. The final limitation of this work is that their contribution focuses exclusively on mobile applications, and the authors do not generalize it to other types of systems.

A more comprehensible and complete formalism has been proposed in recent work by Bucchiarone and Mongiello [23], in which the authors introduce a formal framework to characterize different aspects of an ensemble-based software engineering. Concretely, they present 1) how to model dynamic software ensembles using typed graph grammar, 2) how to specialize and re-configure ensembles, and 3) how to manage collective adaptations in an ensemble. As part of this work, the authors use Typed Graph Grammars in combination with Labelled Transition Systems to formally define system context, context awareness, and system adaptation in the frame of system ensembles. According to the authors, the adaptation should consider 1) the goal of some context properties that adaptation should achieve, 2) the system context at the moment that the need for adaptation is triggered, and 3) the functionalities of the cells (the systems) present in the system (the ensembles) at the moment that the need for adaptation is triggered. Although it is not an extensive and complete framing like the one we propose as part of this dissertation, still the work by Bucchiarone and Mongiello [23] acknowledges the importance of considering the

functionality that adapts, the quality and the context as inseparable notions from system adaptation. The aspects of uncertainty, the separation between adaptation and business goals, the fact that adaptation is always in relation to the satisfaction of the adaptation goals within a specific range of values, and in overall, what this all implies for engineering self-adaptive systems have not been considered as part of the definitions in [23]. However, probably the biggest limitation of this work—similarly to some of the other existing works described above—is that the authors provide formalisms for adaptation exclusively as an emerging property from system ensembles/system collaboration. Nonetheless, the notion of system adaptation needs to be considered and defined in independence from system collaboration, multi-agent systems and ensembles.

Summary. Despite the growing and active community for the past two decades, including the expanding interest in self-adaptive systems in the literature of software engineering, there is a sparsity of works that focus on formal definitions of system adaptation and self-adaptive systems and characterizations of self-adaptive systems. It is notable that none of the existing works which propose some formalized world of concepts—which we analyzed above—have the complete framing and consider all the necessary aspects to understand system adaptation and, respectively, self-adaptive systems, resulting in a lack of a holistic, comprehensible and precise formal definition of these terms.

Through our literature analysis in this section, we have also detected a common pattern among the existing works that aim to propose formal efforts to define (self-)adaptive systems [129, 120, 8, 23]. Probably done to compensate for the lack of a better understanding of system adaptation, these works establish their definitions for self-adaptive systems by exploiting various facets like collaboration, multi-agent systems, decentralization and ensembles. However, in essence, these notions are independent and separate from system adaptation and should not be leveraged as foundations to define self-adaptive systems.

Another unexpected insight from our literature analysis is that the notion of uncertainty has not been considered in any of the formal efforts to define self-adaptive systems, although uncertainty is considered the main reason for self-adaptive systems in every published paper on this topic, as well as in the majority of the informal definitions of these systems.

With exception to the last paper by Bucchiarone and Mongiello [23], none of the formal efforts to define self-adaptive systems tries to define adaptivity as a property of the systems first before proceeding to define self-adaptive systems.

And lastly, all of the papers summarized in this section entirely exclude the aspect that a self-adaptive system should evaluate itself regarding the fulfilment of its mission, based on which the system changes its own behaviour when the evaluation shows that the system is not fulfilling its objectives. These ideas were initially positioned by Laddaga in his informal definition of self-adaptive software; however, they were completely abandoned in the other works in the literature after the MAPE-K closed feedback loop for engineering self-* systems was published. Hence, in our opinion, they are essential to be considered while defining self-adaptive systems, together with the definition of system adaptation.

8.1.3 Other notions related to system adaptation

Across the literature, self-adaptation has been closely related and even used interchangeably with other terms like autonomy, context-awareness and self-awareness. This section summarises some of the existing efforts to define these three notions.

Although the interest in autonomous systems has become more prominent in recent years through the emerging high-profile application such as autonomous cars, the initial efforts in the literature to formally define autonomy as a system property were made in 1995. In [83], Luck and d'Inverno argue that the terms agency and autonomy are often used interchangeably without considering their relevance and significance, and in response, they propose a three-tiered theory using the Z specification language. In their three-tiered hierarchy, the authors distinguish between objects, agents, and autonomous agents. Concretely, in their definition of autonomy, as a focal point, the authors introduce *motivations*—"higher-level non-derivative components related to goals." Namely, according to their definition, autonomous agents have certain motivations and some potential to evaluate their own behaviour in terms of their environment and their respective motivations. The authors further add that the behaviour of the autonomous agent is strongly determined by and dependent on different internal and environmental factors. Although the authors acknowledge the importance of considering different internal and environmental (i. e., contextual) factors while defining autonomy, in their formalisms, the importance of the user in defining autonomy is entirely omitted. On the contrary, in our work, we explicitly consider the user and how the user's involvement in the system's operation decreases proportionally to increasing the system's autonomy. We define levels of system autonomy by focusing on the system's function and how much from the user's logic is "shifted" to the system in the higher levels of autonomy.

Barber and Martin motivate their work in [10] based on the fact that there is little agreement about the concept of autonomy in the literature, stating that a formal definition of agent autonomy is necessary to provide a foundation for work in this area as well as to support the operational deployment of the concept. As part of their contribution, they aim to propose a framework for interpreting agent autonomy. They conclude that there are three key elements necessary to model agent autonomy: (1) the identification of the goal around which the autonomy assessment is focused, (2) specification of the problem-solving role for each decision-making agent, and (3) explicit declaration of which agents are required to carry out the decisions made by the decision-makers.

Gunderson and Gunderson in [51] focus on the terms intelligence, autonomy and capability. The authors argue that although these three terms are often used in the literature interchangeably, they are far from equivalent. They also state that in order to have a common framework for discussion, it is necessary to have at least a working definition of these terms, additionally emphasizing that having more precise definitions than the working definitions (e. g., formal definitions) would be better preferred. With this, the authors of [51] once again validate the importance of terminological clarity in engineering, which is crucial not only for discussions but the overall engineering processes, which was the main motivation for defining both system autonomy, as well as system (self-)adaptation to which we contribute with this dissertation.

However, as part of this work, the authors ended up putting the primary focus on intelligence and capability, defining intelligence as a cognitive process that allows a system to propose a viable solution to a problem or task and capability as the ability to implement or execute a proposed solution in a dynamic, uncertain environment successfully. In relation to defining autonomy, they state that if a system should be tightly controlled, then it needs to be tightly controlled. This is similar to the differentiation we make in our work, where we argue that not every system function is meant to be autonomous and that autonomy is a property of a concrete function. Moreover, as part of their work, the authors identify the importance of considering the nature of the domain, i. e., the context of the system. Concretely, the autonomy of a system differs if the systems operate in static and deterministic contexts versus dynamic and uncertain domains.

Compared to all the prior related efforts, as part of our work, we additionally take into account the importance of considering the *learning* aspects in autonomous systems, especially when 1) the systems operate in highly dynamic, uncertain and unknown environments, and 2) the user's control on the system reduces. Finally, to the best of our knowledge, no prior work defines different levels of autonomy *formally*.

Context awareness and self-awareness in computing systems is another field of research related to system self-adaptation. Elhabbash et al. in [37] view self-awareness as a property for enriching self-adaptive systems similar to other self-* properties (e. g., self-healing, self-organising, etc.). However, they also state that the relation between other self-* properties and self-awareness is not entirely clear, and it presents a future research challenge. Broy in [17] states that context awareness is closely related to adaptation and context modelling, and it is a prerequisite for self-adaptation, to which the author refers to as automatic adaptation. Furthermore, according to Broy, context awareness has two aspects: 1) self-awareness enabling a system to reflect its own system model, its own state, etc., and 2) awareness of its context (i. e., context-awareness). Finally, Petrovska in [94], identifies self-awareness and context-awareness as prerequisites for self-adaptation. Through discussion of the properties of self-adaptive systems, the author depicts the relation between self-awareness and self-adaptation. Furthermore, the author exemplifies and identifies two different levels of awareness in computing systems: *primary system awareness* and *secondary system awareness*. Concretely, every system has some knowledge by design: different models of the system, the system's state, the encoded specifications of the system, or in general, the implementation of the system which provides the system's functionality (i. e., system function (sf), adopted from [75]). This knowledge is considered as an "innate" system awareness that the system function (sf) has by design and comprises the primary system awareness. On the contrary, the reflection of that internal knowledge (the "innate" awareness) with respect to certain adaptation objectives as part of the knowledge of the adaptation logic is called secondary system awareness.

Summary. Similarly, as for self-adaptive systems, there needs to be a more precise definition of autonomous, context-aware and self-aware systems. This results in a blurry and imprecise distinction between these different types of systems and consequently in ambiguous and interleaved usage, as well as vague association among the systems. To the fuzziness in the distinction between these systems additionally contributes the fact that the proposed MAPE-K conceptual model is used not only as a reference model for

engineering self-adaptive systems but as well for engineering self-* systems in general (e. g., self-healing, self-organising, self-aware, etc.). Getting a clear definition of system adaptation and self-adaptive systems and a more precise terminology semantics will also contribute to distinction and differentiation among the other types of self-* systems and self-adaptive systems.

8.1.4 Overall summary

Despite 1) the acknowledgement of adaptation as an emerging property of software systems, 2) the expanding interest in self-adaptive systems in the literature of software engineering, 3) the various systematic mapping studies and literature reviews in the field of self-adaptive systems [90, 84, 74, 119, 101], and 4) many methodological, architectural and technical solutions for engineering self-adaptive systems, it can be observed that there is a lack of contributions on the foundations of these systems, which aim to understand and define the essence of self-adaptive systems. Especially there is a clear gap in the literature when it comes to formal definitions, as well as formal characterizations of self-adaptive systems. This also would include conducting research on defining system adaptation, which is a prerequisite for a subsequent definition of self-adaptive systems.

The lack of precise understanding of *what* are self-adaptive systems has different software engineering consequences and implications, for instance, *how* to build or engineer these systems that go beyond the famous MAPE-K conceptual model (further discussed and analyzed in Section 8.2). A more specific semantics accompanying the MAPE-K reference model will also enable a better separation and characterization of, for example, self-adaptive and self-organizing or self-aware systems. And finally, having better semantics of the notion of self-adaptive systems will 1) complement the already existing works in this field, and 2) set the foundation on *how* to evaluate and compare these systems in the future.

Gap 1: The literature in software engineering lacks a precise, comprehensive and broadly accepted formal definition of system adaptation and self-adaptive systems.

8.2 Engineering Self-Adaptive Systems

The lack of precise definition of self-adaptive systems has various software engineering consequences and implications, for instance, *how* to build these systems. To date, several approaches have been proposed in the literature to facilitate the engineering of self-adaptive systems. However, all of them based on authors' intuitive understanding of these systems. In this section, we examine the most relevant approaches with respect to our proposed logical architecture proposed as part of this thesis.

8.2.1 Models

More than twenty years ago, Kephart and Chess, in their well known IBM manifesto on autonomic computing have proposed the peculiar ideas for autonomic computing systems that "can manage themselves given high-level objectives from administrators" [66]. The envisioned autonomic systems organise and manage themselves in a completely

autonomous manner. For these systems, the designers and the engineers have become obsolete and are replaced by human administrators or end-users that merely specify the system's high-level business objectives and do not deal with low-level technical details. From the current time point, it is very challenging to argue how these systems will perform and how they will be engineered. As a result, an evolutionary, iterative approach toward understanding, designing, and engineering succeeding systems should allow continuous step-by-step integration of the contemporary concepts and ideas. We consider the *self-adaptive systems* as an intermediate step—an iteration, which, once fully understood, brings us a step closer to fully autonomous systems.

Besides setting the vision and the foundation for a whole new research domain, in our view the most significant contribution of the autonomic manifesto is the proposed *MAPE-K conceptual model* [66]. As explained previously in Chapter 1, most of the architecture-based self-adaptation systems are built upon the MAPE-K loop, or at least on different variations of the five phases of the MAPE-K. However, the MAPE-K is not helpful in designing an actual technical implementation of a self-adaptive system since, as discussed in different sections of this thesis, it is unclear how self-adaptive systems engineered according to the MAPE-K differ from the "ordinary", non-adaptive systems. Furthermore, MAPE-K is used not only as a reference model for engineering self-adaptive systems but also for engineering self-* systems in general. Therefore, it needs to be clarified how a self-adaptive system engineered according to the MAPE-K conceptual model differs, for instance, from a self-aware system, since the same conceptual model is used for engineering self-adaptive, self-aware and all the self-* systems. Various limitations and flaws of the MAPE-K conceptual model served as a backbone for eliciting different research problems throughout this dissertation. In the following, we again briefly summarize some of the limitations of the MAPE-K.

Although MAPE-K gives some intuition behind the engineering of self-adaptive systems, primarily by the separation of concerns between the managed element and the adaptation logic, a more specific semantics of these two components within the conceptual model is still lacking; therefore, it remains open for a subjective interpretation of the designer or the engineer of the system. This is not necessarily a disadvantage, but it opens a space for misinterpretation. Concretely, it can be interpreted that the managed element, which is just any system, already contains elements of monitoring, analysis, planning[1], and execution; as a result, blurring lines between the managed element and the adaptation logic. As a result, this enables every system (e. g., CPS) to be misleadingly labelled as self-adaptive. This separation becomes additionally blurry with the introduction of different MAPE-based patterns for self-adaptive systems [123, 101] that focus on different combinations of decentralization of the four MAPE phases. Although different MAPE-based patterns could be more informative regarding the system's design, inherently, they have the same limitations as the MAPE-K closed feedback loop itself: their high level of abstraction, without providing any characterization of how a system built upon the MAPE loop differentiates from the "ordinary", non-adaptive systems, nor a minimal set of

[1]It is also worth mentioning that in the research domain, the differentiation between the analysis and planning phases has been argumentative.

requirements in order for the system to be considered as a self-adaptive.

Moreover, the aspect that a self-adaptive system evaluates its own behaviour and changes behaviour when the evaluation indicates that the system is not accomplishing what it is intended to do, as envisioned by Laddaga in his informal definition of self-adaptive systems [77], is entirely left out in the MAPE-K conceptual model.

Summary. The conceptual MAPE-K model as a reference model for engineering self-* systems has a very high level of abstraction, which is not necessarily helpful in designing an actual technical implementation of a self-adaptive system. Furthermore, MAPE-K does not provide any insights into how self-adaptive systems (engineered according to MAPE-K) differ from ordinary systems, which are considered non-adaptive, or differ from other self-* systems.

8.2.2 Patterns

Weyns et al. [123] propose patterns for decentralised MAPE[2] control loops in self-adaptive systems. The authors argue that when systems are large, complex and heterogeneous, a single MAPE loop might be insufficient for managing all the system's adaptations, and the need emerges how to decentralise each of the MAPE phases. A pattern describes a generic solution for a recurring design problem [87]. The authors in [123] develop a systematic approach for describing multiple interacting MAPE loops, based on which they propose five patterns for MAPE decentralised control in self-adaptive systems. The patterns proposed in [123] are derived from common knowledge in the field of self-adaptation and the authors' experiences with building self-adaptive systems. Although the different patterns for decentralisation of the MAPE loops can foster more structured formations for engineering self-adaptive systems, the shortcomings, as discussed in the previous section, remain the same as the MAPE-K loop itself. Concretely, patterns remain on a very high level of abstraction, and they are not necessarily beneficial for the design of an actual physical architecture of a self-adaptive system. It is also essential to point out that the proposed design patterns completely exclude the knowledge component.

In [91], Musil et al. report the results of a systematic survey on CPSs that combine different self-adaptation mechanisms across the technological stack of the system. Foreseeable, their results showed that the majority of the studies for engineering self-adaptive CPS combine variations of the MAPE closed feedback loop. Additionally, as part of this work, based on a few studies from their survey, the authors identify three adaptation patterns with different combinations of multiple types of self-adaptation within the system. Their patterns distinguish between the type of adaptation mechanisms, their layer locations, and the cross-layer inter-adaptation interactions between the respective mechanisms. The three proposed patterns aim to tackle the following three problems: 1) a distributed application seeks to improve the utility of its services to the physical resources by dynamically exploiting rich context information, 2) a distributed application exploits data of individual resources to improve its overall utility by changing the resource configuration that produces the functionality of the application, and 3) a distributed application seeks to improve the

[2]Note that the Knowledge component (K) is not explicitly considered as part of the proposed patterns.

overall utility of its service, which requires the autonomous entities to efficiently share information and coordinate their tasks on a local basis.

In their study, the authors have explored how different adaptation mechanisms have been utilised for self-adaptation in CPS, without providing a common and shared understanding of what system adaptation means in the first place, nor did they define self-adaptive systems. Through their work we can once again show and validate our findings from Chapter 4, that very often works in this domain leverage the notion of decentralization or distribution to compensate for the lack of clarify of the core aspects of adaptation and self-adaptive systems. Furthermore, the patterns identified in this work are not informative for the specific design of a self-adaptive CPS, nor do they tackle concrete specificities and properties of the CPSs that are unique for these systems and differ, for example, from other distributed and decentralised applications, or software systems in general.

Summary. The existing works on patterns for engineering self-adaptive systems and, respectively, self-adaptive CPSs, primarily utilize the concept of MAPE (please note that the Knowledge component is excluded) while completely omitting any discussion on system adaptation and why the systems that are built according to their proposed patterns are self-adaptive. Additionally, although one might argue that knowledge is implicitly considered as part of the other four phases of the MAPE, the existing patterns exclude the knowledge component and leave out any emphasis on the importance of the consideration of the knowledge as part of their works. This contradicts our findings in which we argue that the knowledge component built according to the concrete system function that adapts, the adaptation goals and the uncertainties, and that keeps models of the context and/or the system that is relevant for the adaptation, is the minimal requirement for a (passive) self-adaptive system.

8.2.3 Frameworks

The most prominent and broadly accepted framework in the literature for engineering self-adaptive systems is the Rainbow framework, developed by Garlan et al. [44]. Rainbow is a two-layered framework for architecture-based self-adaptation utilising utility theory. In this framework, the adaptation infrastructure is tailored using the system-specific adaptation knowledge, including the types and properties of components, behavioural constraints, and adaptation strategies (to which we refer as actions towards adaptation in this dissertation). Rainbow uses specific mechanisms for planning, which is something that we do not prescribe in our logical architecture, and it is left to the designers of the system. Concretely, utility theory in their approach is used for finding the optimal trade-off between two quality objectives (performance and cost in their used Znn.com exemplar [27]) when the system is exposed to uncertain conditions (unknown periods of peak requests). The adaptation is triggered (i. e., different adaptation strategies are executed—to which we refer actions towards adaptation in our formalisms) based on the desired value of the trade-off in different conditions. Furthermore, the adaptation logic of the framework is predefined during the design of the system, as Rainbow does not support run-time modification of the adaptation logic, which disables the approach to deal with uncertain situations and changing conditions that were not known during the

design of the self-adaptive system. In Appendix A.5, we exemplify further some of our theoretical contributions and terminology that we propose in this dissertation on the Znn.com exemplar.

D'Angelo et al. [30] propose CYber-PHysical dEvelopment Framework (CyPhEF), a framework for developing self-adaptive CPSs based on model-driven engineering paradigm [31]. CyPhEF utilizes models as primary artefacts for engineering self-adaptive CPSs. Specifically, CyPhEF promotes MAPE-K components to first-class modelling abstractions and provides: 1) a Domain-Specific Environment for specifying the MAPE-K control loop model, and 2) a Cyber-Physical Simulation Platform for simulating the designed self-adaptive CPS. In other words, the Domain-Specific Environment allows defining the control architecture of the CPS under consideration. The designed self-adaptive CPS is simulated using the Simulation Platform.

Seiger et al. [108] also propose a framework for engineering self-adaptive CPSs. Their work examines the application of Business Process Management (BPM) and workflow technologies in the new areas of the Internet of Things and CPSs to increase automation, better resource utilization, and higher product quality. The authors present an approach for enabling self-adaptive workflows based on the MAPE-K, to which, in addition to the feedback from the process management system and services, additional sensor data from physical devices is used to monitor and analyze the real-world effects of the process activity executions. This work uses graph queries on a knowledge base to find a compensation (e. g., a replacement resource) to be activated in case of inconsistencies and unanticipated errors.

Although both of these works by D'Angelo et al. [30] and Seiger et al. [108] propose frameworks for engineering self-adaptive CPSs, both the frameworks propose a MAPE-K-based solution and *none* of the frameworks supports run-time modification of the adaptation logic. Furthermore, D'Angelo et al. acknowledge that developing self-adaptation for CPS is more challenging than software systems since a CPS contains both control and physical aspect; however, the authors do not specify the concrete characterization of the CPSs that their proposed framework tackles. The authors continue with the argument that centralized control is not adequate in large distributed settings and that self-adaptation might be achieved through decentralized control. Nonetheless, how decentralizing the control (the MAPE-K control loop in this case) is related to the adaptation capabilities of the systems has not been addressed by the authors. Decentralization and adaptation are two distinct and independent concepts, and the notion of decentralization should not be leveraged to compensate for the imprecise understanding of the notion of adaptivity.

Camara et al. [24] propose a framework for self-adaptation based on quantitative synthesis and verification that separates the planning phase into 1) architecture reconfiguration and 2) task planning problems on a use case of mobile service robots—similar to the use case in this thesis. The authors have identified the problem of the large solution spaces while searching for the best combination of software architecture configuration and task planning specification for adaptation. This is because quantitative guarantees associated with reconfiguration and behaviour strategies depend on different information from various models that change at run-time. This potentially invalidates every quantitative guarantee associated with pre-computed strategies (the core limitation of the MORPH approach [14], see Section 8.2.4). This paper acknowledges the importance of information from

models that change at run-time, and their proposed approach enables automated run-time decision-making for self-adaptation. In our architecture, similarly to the framework by Camara et al., we also promote both behavioural adaptation (modification of the managed elements) and structural adaptation (modification of the adaptation logic). However, the work by Camara et al. [24] has a couple of limitations in comparison to the third contribution of this thesis (i. e., the logical architecture): 1) the authors in [24] mainly focus on the planning phase of the self-adaptive system, and it could be seen as one potential technical solution for the planning component of our logical architecture, 2) our logical architecture is built while considering specific characteristics of single and multi-agent CPSs (e. g., the partiality and the uncertainties of the observations, the dynamic context, the need for reasoning and knowledge aggregation, etc.) that are entirely out of scope in the framework by Camara et al. 3) our architecture is built within the proposed formalisms on system adaptation, consequently supporting a more systematic development of self-adaptive CPSs, and most importantly 4) our logical incorporates the notion of the Quality Function, which enables the self-adaptive system to evaluate itself and indicate when the system is not accomplishing its goals based on which self-adaptation is triggered.

Finally, a few other frameworks have also been proposed in the literature that apply to only a single technology, e. g., Java-based applications [9, 50], or to a single domain, e. g., mobile applications [55, 105]. However, these works not generalize beyond other types or classes of systems, nor answer why they consider the systems built according to their proposed frameworks to be self-adaptive.

Summary. The proposed frameworks in the literature for engineering self-adaptive CPSs [30, 108, 24] do not focus on the specific characteristics of CPSs, according to which the respective solutions are proposed. Moreover, except for Rainbow [44], we can observe a similar pattern as in all the other works in this research domain. Namely, the lack of a more profound understanding of system adaptation and self-adaptive systems has clearly reflected in the proposed frameworks that we analyzed as part of this section. The liberate use of the terminology of self-adaptive systems, including the intuitive understanding of these systems, results in ambiguity and difficulties in comparing the existing frameworks.

8.2.4 Architectures

On the architectural level, Affonso et al. [3] and Braberman et al. [14] propose reference architectures based on reflection for self-adaptive software and a reference architecture for configuration and behavioural self-adaptation, respectively. Reference architectures are considered as reusable artefacts that combine the knowledge of architectures of software systems in specific domains. Reference architectures support the development, standardisation, and evolution of systems of those domains. To the best of our knowledge, these are the only two existing reference architectures for engineering self-adaptive systems.

Affonso et al. [3] present a reflection-based reference architecture for self-adaptive software, named RA4SaS, which aims at developing software entities that are transparently monitored and adapted at run-time. Namely, to perform these operations, this architecture proposes using modules in an “assembly line”, which allows a software entity to be disassembled, adapted, and reassembled automatically by these modules. Through a proposed case

study, the authors show ways to apply structural adaptation of software entities through automatic mechanisms. The theoretical shortcomings of this work is two-fold, which as a consequence reflects on the architectural solution proposed in the paper. First, the authors' understanding of self-adaptive systems as "systems that can adapt by modifying their structure and behaviour" is almost the same as the informal definition by Broy [17], which we extensively discussed and analyzed its shortcomings in Section 1.1.3. Second, probably as a result of the first point, it is not clear how the automated process proposed in the architecture is related to system adaptation and why they refer to their process as adaptation and not simply as automated module change.

Braberman et al. [14] present MORPH, a reference architecture that distinguishes the dependencies between structural reconfigurations and behavioural adaptations in self-adaptive systems. Concretely, the proposed architectural approach involves run-time change of the system configurations (e. g., the system's components and their bindings) and behaviour update (e. g., components orchestration, reactive behaviour, etc.) based on design time predefined configurations. The architecture is structured in three main layers: Goal Management, Strategy Management, and Strategy Enactment, which control the target system. The target system provides the system functionality, similarly as the managed element in our work. In MORPH, the authors consider every layer as implementing a separate MAPE-K loop. Orthogonal to the three layers, the architecture contains the Common Knowledge Repository, which keeps the goal model of the system specified by the system's developers. The goal model comprises assumptions about the system state, the system goals and the environment and informs the design of the three layers of the reference architectures. This is similar to our work (see Chapter 5), where the goals (business and adaptation) inform the design of the Quality Function and the adaptation logic. Similarly, in our work we say that the knowledge in the adaptation logic keeps model of the states of the system and the context, relevant for the concrete adaptation. In our logical architecture, compared to the both reference architectures described above, the adaptation logic not only adapts the managed element(s) but also modifies itself during run-time. Although it can be noted that in MORPH the authors have some better understanding of the semantics of self-adaptive systems, none of the existing architectures is outlined within a formal framework that defines what means for a system to adapt at first place, and differentiates between self-adaptive and ordinary, non-adaptive systems.

Summary. Our literature analysis showed a scarcity of architectures for engineering self-adaptive systems in the literature. This is presumably because there is a lack of understanding and definitions, and in more general, semantic clarity about what self-adaptive systems are, which is a necessary first step before discussing and proposing approaches for engineering these systems. The two existing reference architectures focus on engineering self-adaptive software systems and utilize the notions of system reflection and reconfiguration. These two concepts could be indeed intuitively related to adaptation; however, in both of these works remains open to what is precisely understood by adaptation in their contributions. Also, both of the reference architectures provide solutions for self-adaptive software systems and not CPSs that operate in dynamic and changing contexts. Finally, in none of the architectures does the adaptation logic gets modified during run-time, nor do they incorporate the aspect of the system evaluating itself based on which the

system changes its behaviour.

8.2.5 Overall summary

To the best of our knowledge, there is no existing architecture in the literature that can serve as a blueprint for engineering self-adaptive CPSs. Concretely, engineering autonomous and decentralised self-adaptive CPSs that make partial and uncertain observations of the dynamic context in which they operate. Moreover, similar to all the other works in the literature in this field of research, there is a lack of a more profound understanding of system adaptation and self-adaptive systems beyond the authors' intuition, which is clearly reflected in the proposed approaches that we analyzed as part of this section. Since there is a lack of formal definitions of system adaptation and self-adaptive systems, none of the existing approaches for engineering self-adaptive systems has been built within a previously existing formal and theoretical framework.

Additionally, in this dissertation, we focus on engineering autonomous and decentralised self-adaptive CPSs that operate in a dynamic context that changes unpredictably during run-time. In response, the knowledge in the adaptation logic that encodes certain aspects of the context (or the system) relevant for the adaptation should be updated during run-time to reflect the actual state of the context and the system during the operation of the self-adaptive system. Therefore, an architecture for engineering this type of self-adaptive CPSs should put particular emphasis on run-time knowledge derivation in the adaptation logic, which enables the adaptation logic to be updated during run-time.

Lastly, none of the existing works incorporates the idea from Laddaga's informal definition of self-adaptive software: that these systems can evaluate themselves, i. e. their own behaviour, based on which they get an indication if they are doing what they are intended to do, i. e. if the systems fulfil their goals or not. When the (adaptation and the business) goals are not fulfilled, this triggers the self-adaptation.

Gap 2: The existing works in the literature do not provide concrete architectures, frameworks nor methodologies for engineering self-adaptive systems, particularly decentralized and autonomous (MA-)SACPSs that operate in changing and uncertain contexts that are only partially observable by the CPSs.

8.3 Mitigating Uncertainties in Self-Adaptive Systems

As part of this section, we examine prior efforts that have contributed towards mitigating uncertainties in self-adaptive systems. In this section, we first summarize the related papers on this topic and then discuss their limitations, which leads to identifying the respective gap.

Whittle et al. [124] propose RELAX—a requirements specification language designed to explicitly consider uncertainty in the specification of the behaviour of dynamically adaptive systems. Dynamically adaptive systems need to operate correctly in a range of environmental and contextual conditions. The nature of these conditions in the respective application domains is often imperfectly understood and, therefore, inherently uncertain. The proposed RELAX language supports explicit expressions of environmental uncertainty

in the requirements of the self-adaptive system, and the vocabulary of RELAX enables identifying requirements that may be relaxed at run-time when the environment changes. RELAX is specified in the form of structured natural language with Boolean expressions, and the semantics of RELAX is defined in terms of temporal fuzzy logic. Furthermore, RELAX has three types of operators to handle uncertainty: temporal, ordinal and modal, and considers four uncertainty factors that warrant a relaxation of the requirements: properties from the environment, properties that the systems can monitor, properties stemming from the relationship between the environment and the monitoring, and finally, the dependencies between the (relaxed and invariant) requirement.

RELAX [124] has also been used as part of the goal-based modelling approach proposed by Cheng et al. [26]. Similarly as in [124], Cheng et al. in [26] emphasize the uncertainty of the execution environment as the motivation for dynamic adaptation. In this work, the authors introduce a goal-based modelling approach based on which the requirements for dynamically adaptive systems are developed. The proposed modelling approach explicitly integrates the uncertainties in the process and in the resulting requirements. Concretely, the approach systematically models the requirements of dynamically adaptive systems based on a variation of threat modelling (e. g., [106]) to uncover places in the model where the requirements need to be updated to support adaptation, resulting in a range of tactics for adaptation which deal with uncertainties. The tactics include 1) adding low-level goals, 2) RELAXing requirements to express bounded uncertainty to accomplish a partial but still suitable satisfaction of the goals, and 3) identifying new goals to mitigate the uncertainty that leads to a new target system.

In contrast to the works by Whittle et al. [124] and Cheng et al. [26] that focus on addressing uncertainties issues in the specifications of the requirements of self-adaptive systems, the following works from Bencomo et al. [11], Esfahani et al. [39] and Filieri et al. [40] focus on tackling uncertainties in making adaptation decisions.

Bencomo et al. [11] propose an approach that uses dynamic decision networks (DDNs) for making decisions under uncertainty in self-adaptive systems. Similarly, as in our work, Bencomo et al. [11] acknowledge that self-adaptive systems must satisfy functional requirements while fulfilling some quality attributes or non-functional requirements (NFRs). The authors state that NFRs play a focal role in choosing between various configurations that drive the run-time adaptation and that the decision on the configuration is made based on utility functions that enable evaluating different trade-offs between the NFRs. Measuring the satisfaction of the NFRs is difficult due to their vague nature. In response, the authors utilize dynamic decision networks (DDNs)—a probability-based technique, to reason about the fulfilment of the NFRs, which later supports the decision-making process of the self-adaptive system. Although various sources of uncertainty have been identified in the literature of self-adaptive systems over the years, in this paper, the authors focus on the uncertainty associated with the satisfaction of NFRs. Their approach can be applied whenever a piece of new evidence is collected for specific events, which might result in a trigger for the adaptation, i. e., a reconfiguration of the system.

Esfahani et al. [39] propose POssIbilistic SElfaDaptation (POISED), a quantitative approach for tackling the challenge posed by uncertainty in making adaptation decisions. POISED builds on possibility theory to assess the positive and the negative consequences

of uncertainty. In this approach, uncertainty estimates are incorporated in the probabilistic analysis of the adaptation choices. The probabilistic analysis uses the principles of fuzzy mathematics [76] and provides a sound basis for representing uncertainty. POISED enables making adaptation decisions that result in the best range of potential behaviour of the self-adaptive system. Contrary to the other approaches that focus on external or environmental uncertainties, this work focuses on uncertainty that is rooted in the difficulty of determining the impact of adaptation on the system's quality objectives, e. g., determining the impact of replacing a software component on the system's responsiveness, battery usage, etc.

Filieri et al. [40] propose KAMI—an approach for verification of non-functional requirements of adaptive software. The authors specifically focus on performance and reliability, which strongly depend on external factors that occur in the environment, which are hard to be understood or anticipated when an application is initially designed. Similarly, even if the predictions are initially accurate, they would probably change continuously during the execution of the application. To express non-functional requirements, the innate uncertainty that characterizes those requirements and, respectively, the systems need to be taken into account. Uncertainty, as such, could be formalized and quantified within the frame of probability. As part of this work, the authors focus on the following two probabilistic Markov models of the application: Discrete-Time Markov Chains (DTMSs) and Continuous-Time Markov Chains (CTMCs) to target the challenge of making adaptation decisions under uncertainty. The authors illustrate their approach on a simplified e-commerce application.

All of the five approaches [124, 26, 11, 39, 40] that we analyzed above identify the concrete type or source of uncertainties that the authors aim to mitigate with their solutions. Also, all of these works provide a concrete technique or method based on which the mitigation is achieved. However, there are a couple of aspects that the prior, related work has not considered, and we summarize in the following. *First*, the prior work primarily focuses on external uncertainties, i. e., uncertainties originating from the unpredictable environment or the system's context. Internal run-time uncertainties, i. e., the systems' sensors, have not been explicitly considered and treated in the existing solutions. *Second*, the existing approaches are often designed for a specific application without providing any further insights into the generalizability of the provided solutions. *Third*, it can be observed that all of the approaches in some way incorporate the MAPE-K feedback loop. However, in none of the approaches, the loop is modified beyond its design time specifications. Namely, the existing works focus either on mitigating uncertainties in the requirements specifications for self-adaptive systems or uncertainties in the decision process of the adaptation actions. However, none of the prior works in the literature focus on mitigating uncertainties in the reasoning process based on which the knowledge in the adaptation logic is consequently updated. This allows the adaptation logic to also change during the system's run-time, enabling the adaptation logic to reflect the current (i. e., the run-time state) of the context and the systems.

It is worth mentioning that in the literature, there are a few more existing works (e. g., [92, 21]) that claim to propose uncertainty mitigation strategies based on the fact that their solutions are simply built upon one or multiple feedback loops (which in essence are nothing else than variations of the closed MAPE-K feedback loop). As explained previously, a

simple feedback loop performs activities like collecting and analysing information, based on which actions are decided and later executed. Hence, the authors of these works [92, 21] interpret that this process intrinsically mitigates the effect of the uncertainties, implying that every approach built upon a closed feedback loop, by default, mitigates uncertainties. Predictably, these uncertainty mitigation approaches do not provide any detail on the type or the range of uncertainty they concretely mitigate (which is necessary while discussing self-adaptive systems, as we previously discussed in the summary of the theoretical findings in Section 1.3), nor on the concrete technique or method based on which the uncertainty mitigation is achieved. Instead, as previously said, they assume that a feedback loop inherently mitigates uncertainties by default. In the previous sections of this chapter and Chapter 1, we have already extensively elaborated on the limitations of the MAPE-K and the various drawbacks which primarily emerge from the lack of the semantic clarity of the feedback loop. The argument that every closed feedback loop mitigates uncertainties by default is another emerging drawback stemming from the lack of semantic clarity.

8.3.1 Overall summary

In architecture-based self-adaptation, almost all the existing uncertainties mitigation strategies are based upon one or multiple feedback loops, i. e., MAPE-K loops. There are works in the literature that assume that their solutions, by default, mitigate uncertainties if their solutions are built and utilise closed feedback loops. However, those works do not even disclose any characteristics related to the nature of the uncertainties that they focus on, including the type or the range of uncertainties that the authors concretely aim to mitigate with their proposed solutions. On the contrary, all the papers that we chose for analysing as related work [124, 26, 11, 39, 40] treat the aspect of uncertainty more carefully. Concretely, each of these works explains and identifies the type of uncertainty that their solution tackles, the origin of the uncertainty, and how it manifests in the system. Most of these works focus on mitigating external uncertainty stemming from the unpredictable environment (the context) in which the systems operate and do not focus on internal uncertainties, i. e., various sensor uncertainties. Additionally, none of the previous works aimed at proposing a domain- and system-independent approach for modelling the knowledge in the adaptation logic and representing uncertainties, which sets the foundation for reasoning under uncertainties based on which the knowledge is updated during run-time to accurately reflect the actual, real-time state of the system and the context relevant for the adaptation. In sum, all the existing works on mitigating uncertainties in self-adaptive systems either focus on mitigating uncertainties in the requirements specifications or uncertainties in the decision process for the adaptation actions. To the best of our knowledge, no previous work focused on resolving external and internal uncertainties in the reasoning process based on which the knowledge in the adaptation logic is updated at run-time.

Gap 3: There is a scarcity of approaches proposed in the literature that allow domain- and system-independent modelling of the knowledge in the adaptation logic and run-time reasoning, based on which the knowledge is continuously updated to accurately reflect the current state of the relevant aspects from the context and the system for the concrete

adaptation.

8.4 Summary of the Gaps

In the following, we summarize all the gaps identified in this chapter. Each gap was thoroughly elaborated in its corresponding section above.

Gap 1. The literature in software engineering lacks a precise, comprehensive and broadly accepted formal definition of system adaptation and self-adaptive systems.

Gap 2. The existing works in the literature do not provide concrete architectures, frameworks nor methodologies for engineering self-adaptive systems, particularly decentralized and autonomous (MA-)SACPSs that operate in changing and uncertain contexts that are only partially observable by the CPSs.

Gap 3. There is a scarcity of approaches proposed in the literature that allow domain- and system-independent modelling of the knowledge in the adaptation logic and run-time reasoning, based on which the knowledge is continuously updated to accurately reflect the current state of the relevant aspects from the context and the system for the concrete adaptation.

9 Conclusion

This chapter concludes the work in this thesis. It first gives an overview of this dissertation and summarizes the contributions. Afterwards, we discuss the limitations and the lessons learned. Finally, we conclude this section with the potential future work.

The term self-adaptive software was noted in the literature for the first time in 1997 in a DAPRA technical report written by Robert Laddaga from MIT [77]. According to Laddaga, "we seek a new basis for making software adaptive, that does not require specific adaptive techniques, such as neural networks or genetic programming, but instead relies on software informed about its mission and about its construction and behavior." Also, in Laddaga's vision, "self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible".

The idea of self-adaptive systems, and in general self-* systems, was popularized a couple of years later, with the publishing of the famous paper on *The vision of autonomic computing* by Kephart and Chess from IBM, also known as IBM manifesto in the literature [66]. In the manifesto, the authors introduce the term autonomic systems as "computing systems that can manage themselves given high-level objectives from administrators" in which the human administrators or end users merely specify the system's high-level business objectives and they do not deal with the low-level technical details of how those objectives are performed, since these tasks are performed by the autonomic system itself.

The ideas positioned in the manifesto, anticipating autonomic, self-engineering, and self-managing systems, remain "ideas that are not science fiction, but elements of a grand challenge" [66]. From today's perspective, it is very challenging to argue how these systems in the future would perform, what capabilities they will possess, and even how they will be engineered. As a result, an evolutionary, iterative approach toward understanding, designing, and engineering succeeding systems should allow continuous step-by-step integration of contemporary engineering practices. Consequently, we consider *self-adaptive systems* as an intermediate step—an iteration, which, once better understood, brings us a step closer to systems with higher autonomy. Considering current engineering practices and state-of-the-art knowledge, we can say that the observed growing trend and interest in self-adaptive systems stems from the quest for increased system autonomy and the continuous engineering aim to decouple the user (specifically the user's control) from the systems.

As discussed throughout this dissertation, the manifesto on the autonomic computing, including the MAPE-K conceptual model for engineering self-* systems as the main contribution of that work, made a large impact in the literature and set a foundation for a whole new research field. However, Jeffery Kephart, who is one of the co-authors of this paper, has recently made an open statement that despite the enormous success of this publication, this work has a *fatal flaw* [65]. Concretely, he referred to the underspecification

of the notion of goals in their vision and the emerging research problems related to it, to which Kephart has dedicated most of his research career after the publication of the autonomic manifesto. Kephart has argued that research needs to pay more attention to the issue of goal specification: what form do goals take, who sets them, how and when are they elicited, etc.

The limitation that Kephart has emphasized in [65] is something we have also identified while working on this dissertation. However, in this dissertation, we identified and primarily focused on *two other major flaws* in the contribution of the autonomic manifesto. The first major flaw that we tackled is how self-adaptive systems engineered according to the MAPE-K closed feedback loop differ from the "ordinary", non-adaptive systems that already incorporate some notion of monitoring, analysis, planning, and executing (e. g., CPSs). The lack of semantic clarity of the terminology has led to a highly ambiguous usage of the notion of system adaptation, resulting in a liberate usage of the term self-adaptive systems without a shared understanding of these systems. Moreover, this issue became more eminent over time with the increased number of publications in this research domain in recent years, in combination with the lack of foundational clarity of these systems. It can be easily observed in the publishing venues in this research field that although there is some intuition behind the terminology—mainly by the more senior researchers—there is no consensus on the definition of self-adaptive systems. This very often results in situations where every system could be labeled as a self-adaptive, which deteriorates the focus from the real research problems of the field. For example, the research in this domain is hardly putting any focus on answering essential questions about self-adaptive systems (e. g., what are these systems, and how do they differ from the "ordinary") and instead focuses on heavily publishing different MAPE-based systems, which—as discussed—opens space for every system to be labeled as a self-adaptive. Focal to defining self-adaptive systems is defining what it means for a system to adapt and, consequently, in order to address the above-identified problems, how adaptation differs from a nominal system function. With this dissertation, we have substantially contributed to closing this large research problem.

The second major flaw from the autonomic manifesto that we tackled in our work is related to the second part of Laddaga's informal definition of self-adaptive software. In his definition, he emphasizes that self-adaptive software evaluates its own behavior and changes it when the evaluation indicates that the system is not accomplishing what it intended. Concretely, the ideas that Laddaga positioned that self-adaptive systems evaluate themselves regarding the fulfillment of certain goals are entirely overlooked by Kephart and Chess in their proposed MAPE-K conceptual model.

To some degree, this could be related to the point that Kephart made in [65], that in the manifesto, they did not put enough attention to the issues of goals specifications, especially since, as we discussed and concluded in our work, the system's adaptation is always in relation to the fulfillment of some (adaptation) goals. In this thesis, to measure the system adaptation, we proposed a metric to which we referred to as *Quality Function*, which quantifies the fulfillment of the adaptation as well as the business goals and enables the self-adaptive system to evaluate its own behavior and to indicate when the system is not accomplishing its goals, which aligns with the second part of Laddaga's definition. Through our formalisms and contributions, we have independently reached to conclusions

and findings that support how Laddaga originally envisioned these systems and whose vision, unfortunately, was affected and got derailed after the autonomic manifesto was published. Increasing the clarity of the terminology will ideally complement various existing approaches for engineering self-adaptive systems and open new directions of research in the future, enabling the community to endeavor to a fuller extent.

9.1 Thesis overview and summary of the contributions

The very initial objective of this thesis was to focus on conducting research on specific aspects of the engineering of self-adaptive CPSs. However, to build self-adaptive systems, we first need to answer what these systems are. Therefore, during the course of this thesis, we realized that there is a more profound research challenge that went beyond the initial vision for this dissertation: *the need to define self-adaptive systems*. Our newly gained insights into the complexity of the problem at hand went a step further since, with time, we also concluded that it is impossible to define self-adaptive systems without first defining what it means for a system to adapt. Unfortunately, defining the notion of *adaptation* is an independent problem on its own, and as such, it transcends the field of engineering and computer science. As a result, with out growing understanding of the research domain and the current state-of-the-art, the focus of this dissertation also continuously expanded from engineering self-adaptive CPSs (as initially intended) to defining system adaptation and self-adaptive systems.

In response, the overarching goal of this doctoral dissertation was two-fold. The first goal was to formulate and define system adaptation and self-adaptive systems. To attain this goal, we conducted fundamental or foundational research. In the second goal, we focused on engineering a specific class of self-adaptive CPSs, for which we employed applied and empirical research. To attain both of these goals, this thesis was constituted of two complementary parts: Part 1—Theoretical foundations for system adaptation and self-adaptive systems, and Part 2—On engineering self-adaptive CPSs in a dynamic context.

During our efforts to define self-adaptive systems, we realized that gaining a more profound understanding of system autonomy is necessary before scoping and discussing self-adaptive systems. In response, in the first part of this dissertation in Chapter 3, we proposed taxonomy and formally defined different levels of autonomous systems. As part of this work, we did not investigate further the relationship between the different levels of autonomy and self-adaptive systems. However, we closed this gap with the rest of the contributions in the dissertation. In this work, to define system autonomy, we differentiated between the system itself, the end-user of the system, and the context in which the system operates, which we also considered as separate entities in our architecture and formalisms. In our definition of system autonomy and different autonomy levels, we 1) put the system function in the main focus, treating autonomy as a property of an individual function, and 2) investigated the degree of interaction between the end-user and the system (i. e., the system function), and 3) emphasized the importance of considering the *learning* aspects in autonomous systems, especially when the systems operate in highly dynamic, uncertain, and unknown environments and when the user's control of the system reduces. As part of

this work, we used the FOCUS [18] formal modeling notation. This work served as a basis for our definition of system adaptation and self-adaptive systems later in the thesis.

In Chapter 4, we conducted a systematic literature review in which we aimed 1) to get a better understanding of the research field and the previous works that aimed to define self-adaptive systems, 2) to qualitatively analyse the existing works in order to give an overview of how these systems have been previously formally defined, and why none of the existing definitions has been more broadly accepted in the community, and finally, 3) we discussed the limitations of the existing works, which provided a better insight of the shortcomings of the prior efforts. Our results showed that despite the growing interest in self-adaptive systems, *only nine papers* out of more than a thousand papers that we analysed as part of the initial set of this study aimed to define self-adaptive systems formally. With our results, we demonstrated our initial observation that most of the papers in the literature use the notion of self-adaptive systems only by intuition. Concretely, more than half of the relevant papers in our study (175 out of 314 papers) did not provide even an informal definition as part of their works. Furthermore, out of the nine primary studies that we thoroughly analysed, only one paper defined *system adaptation*; however, only within the frame of system ensembles. Our results further showed that roughly half of the nine primary studies provide their formalisms on self-adaptive systems by taking advantage of some other concepts, e. g., ensembles, collaboration, distribution and decentralisation. Based on our analysis, we elicited requirements for a holistic and formal definition of self-adaptive systems. To summarise our results and findings: 1) defining self-adaptive systems is not possible without defining system adaptation first, 2) a formal definition of self-adaptive systems should systematically consider different characteristics of self-adaptive systems in its formalism, in particular the aspect of uncertainty, and 3) system adaptation is not an emerging property from collaboration or decentralisation, and it should be considered, understood and defined in independence from these concepts.

Based on the elicited requirements for defining self-adaptive systems, in Chapter 5, we first formally defined system adaptation and proposed the SACTC framing required for specifying system adaptation. SACTC is an acronym that stands for system function, adaptation goals, conditions, time period and convergence parameters. Namely, we defined system adaptation by separating it from the system function, which is the first premise in the differentiation between self-adaptive and the "ordinary systems", considered as non-adaptive. Similarly as in Chapter 3, we treat adaptation as a property of a single system function. However, by this we do not mean a simple function, and the granularity of the function matters. In sum, to answer if a system (self-)adapts, we first need to identify what system function is considered as adaptive. This is further supported by the separation between business and adaptation goals. The business goals are related to the functional requirements of the system (i. e., the system function) that gains the adaptation capabilities—the managed element as part of a self-adaptive system. On the other side, the adaptation goals are one or more quality objectives, and they are concerns of the adaptation logic. The overall goal of a self-adaptive system is to maintain the fulfilment of the business goals, while preserving or even improving the adaptation goals, despite the dynamic and uncertain internal and external conditions. In response, the adaptation logic of a self-adaptive system is constructed considering 1) the specific adaptation goals

and 2) the specific internal (system) and external (contextual) conditions or uncertainties, according to which the system (i. e., the system function) adapts. Therefore, the same system (i. e., system function or managed element) adapts differently to different adaptation logics, which are built according to different adaptation goals and conditions. This is another premise in the identification of system adapting and system functioning.

As discussed in the previous section, in our work, to measure system adaptation we introduced a metric to which we referred as *Quality Function*, which quantifies the fulfilment of the business and the adaptation goals. We considered the system function as adaptive, if and only if the Quality Function of the system function—in other words the fulfilment of the goals—resides in a specific range of values. Or as expressed in our formalisms, it converges towards a limit ℓ and convergence closeness ε for a concrete SACTC framing specification. The Quality Function that we proposed as part of our work, in the words of Laddaga, provides the foundation for evaluating the behaviour of the (self-)adaptive system and indicates when the system is not accomplishing what is intended to do, i. e., when the goals are not fulfilled. Based on the 1) the informal definition of self-adaptive software by Laddaga and 2) our previous framing and formalisation of system adaptation, as part of in Appendix A we formally defined passive and active self-adaptive systems. As part of the overall theoretical framework in Appendix A, we also complemented the preliminary characterization of self-adaptive systems from Chapter 5 and elicited requirements for self-adaptive systems as such. Lastly, we specified the process of self-adaptation as part of active self-adaptive systems.

In the second part of the thesis, we focused on the engineering of self-adaptive CPSs in dynamic, uncertain and partially observable context. In Chapter 5, we proposed a logical architecture for engineering self-adaptive CPSs. As extensively discussed throughout this dissertation, the proposed MAPE-K conceptual model for engineering self-*, including self-adaptive systems, has several limitations that blurry the lines between self-adaptive systems and the ordinary systems, considered non-adaptive. Additionally, the MAPE-K has a very high level of abstraction and low level of details that—apart from the separation of concerns among the managed system and the adaptation logic—does not provide any constructive insights and semantics on 1) how to engineer self-adaptive systems (e. g., potential technical implementation of a self-adaptive system), and 2) how self-adaptive systems differ from the other self-* systems. In this contribution we embed the formalisms of system adaptation and self-adaptive systems into a logical architecture for engineering self-adaptive CPSs—at an intermediate level of abstraction—that narrows the gap between the formal foundations and potential technical implementation for a class of systems. We focused on decentralized and autonomous (MA-)SACPSs that operate in dynamic, uncertain and partially observable contexts, but also the systems themselves are exposed to various run-time uncertainties primarily originating from their sensors. As part of Chapter 5 we also discussed future research directions emerging from the proposed logical architecture. In our logical architecture, not only that the managed elements (i. e., the CPSs) are adapted, but also the adaptation logic "learns" and updates its knowledge during run-time to accommodate run-time uncertainties emerging from the dynamic and changing context in which the systems operate. This enables the adaptation logic to accurately continue reflecting the run-time state of the context and the system relevant for the concrete

adaptation.

To the best of our knowledge, there are no approaches proposed in the literature that allow domain- and system-independent modelling of the knowledge in the adaptation logic, which also allows 1) capturing various (internal and external) run-time uncertainties and 2) reasoning, based on which the knowledge in the adaptation logic is updated at run-time. In response, in Chapters 6 and 7, we proposed a methodological approach for knowledge and uncertainty representation and reasoning based on which the knowledge in the adaptation logic is updated during run-time to reflect the run-time state of the context and the systems. Our methodology utilizes the theoretical framework of Subjective Logic (SL). We also proposed a model problem from the robotics domain and a ROS-based simulated implementation of the multi-agent robotics system for demonstration and evaluation purposes. The implementation of our robotic system allows simulating various run-time uncertainties, as previously discussed in Section 2.1. Concretely, in Chapter 6, we conceptualized the methodology and implemented the initial version of the robotics system, based on which we conducted the preliminary evaluation. Whereas, in Chapter 7, we propose the fully developed methodological approach for knowledge and uncertainty representation and reasoning at run-time, completed implementation of the ROS-based multi-robot system in which we simulate various run-time uncertainties, and an evaluation through extensive controlled experiments.

Our preliminary results in Chapter 6 indicated that the time to update the knowledge increases when we use the reasoning and aggregation of the observations before updating the knowledge, compared with no knowledge aggregation. However, in the latter, the uncertain and potentially wrong and conflicting observations are directly propagated as knowledge, resulting in a reduction of the system's efficiency (i. e., the fulfilment of the adaptation goals), which, in response, potentionally results in a reduction of the Quality Function. From our preliminary results, we could conclude that there is a trade-off between the accuracy of the knowledge in the adaptation logic and the number of completed tasks. With the additional empirical evaluation that we conducted in Chapter 7, which used an extended implementation of the robotics system, we validated the preliminary results. Nonetheless, to our surprise, the results from our experiments and some additional analysis that we conducted in Chapter 7 revealed various other unexpected insights, which as a result, required an enhancement of the proposed method and approach. Conctetely, our results showed that none of the original SL fusing operators Section 2.3.3 that we initially intended to use as part of our evaluation is capable of long-term, real-time knowledge aggregation. To this point, as another contribution in Chapter 7, we proposed a new operator for fusing SL opinions that addresses the limitations of the original fusing operators proposed by Jøsang [61, 60]. To summarize, in Chapter 7, we evaluated the feasibility of different SL aggregation schemes, the effectiveness of knowledge aggregation with SL, and we did a sensitivity analysis of the impact of the threshold value.

To conclude, in general in the literature, foundational or fundamental research is conducted prior to any solution, and this type of research usually does not generate findings or inventions with an immediate application in practice. However, in this doctoral thesis, we complemented the first part of the thesis, i. e., the theoretical foundations for system adaptation and self-adaptive systems—our fundamental research, with the second part in

which we conducted applied and empirical research for a concrete class of self-adaptive CPSs. This thesis provides a continuous and streamlined engineering process from formal definition to a logical architecture embedded in the proposed theoretical framework to a concrete system implementation from the domain of multi-agent robotics. We achieved this by bridging the various proposed types of contributions and solutions, all of them with different generality and level of abstraction, through a consistent storyline: the theoretical contributions (in Chapter 4 and Appendix A), the architectural contribution (in Chapter 5), and methodological contribution (in Chapters 6 and 7), and finally, the technical contributions (in Chapters 5 to 7).

9.2 Lessons Learned

In this section, we highlight some of the most significant lessons learned as part of this thesis. There is relativity in answering if a system is adaptive and it cannot be determined in isolation without specifying with respect to what it is considered to be adaptive. Concretely, (1) which function of the system is treated as an adaptive, (2) what is the adaptation goal (i. e., the quality objectives) according to which the system (i. e., the system function) adapts, and (3) according to which contextual and internal uncertainties is the system (i. e., the system function) considered to behave adaptively.

In this thesis, we define that a system self-adapts with the intention for adaptation to be eventually achieved. As part of our work we also differentiate between passive and active self-adaptive systems. In the passive self-adaptive systems, the system (i. e., the system function) is enriched with an adaptation logic (in the minimal case an additional knowledge component) constructed in accordance to the system function, the adaptation goals and the uncertainties according to which the system is considered to behave adaptively. Whereas, active self-adaptive systems, to which we also refer to as real self-adaptive systems, extend the passive self-adaptive systems with a feature which enables the system to evaluate itself, i. e., the fulfilment of the (adaptation) goals, or the system adaptation, based on which different actions towards adaptation are chosen.

9.3 Future Work

The advantages of working on fundamental research is the fact that the results and the findings set the foundation for a wide range of new research directions. We outline some of those directions as potential future work in the following.

A taxonomy of self-adaptive systems. Based on the different levels of self-adaptive systems and the distinction between passive and active self-adaptive systems, one potential direction for future work is proposing a taxonomy of self-adaptive systems and mapping some of the existing approaches for engineering self-adaptive systems from the literature to the proposed taxonomy.

Extend our implementation of the selected use case for active self-adaptation.

As discussed previously, the active self-adaptation, in which the value of the Quality Function is used to steer the self-adaptation actively, is out of the scope of the contributions that include the implementation of the robotics use case. As a result, preliminary empirical evidence that supports our theory on the active (or real) self-adaptive systems remained out of the scope of this thesis. This aspect should be considered as an immediate future work.

Introduction of another model problem from practice. Findings of fundamental studies and research are usually applicable to a wide range of use cases and scenarios. To demonstrate (most of) the proposed theoretical concepts in this dissertation, we created a model problem from the robotics domain and implemented a multi-agent ROS-based robotics system. Integration of a second model problem and application could be another future work.

Further analysis of the theoretical contributions of the thesis. Related to the previous point, the theoretical foundation we put as part of this thesis has opened many potential research directions for the future. First, one could conduct more qualitative evaluations of system adaptation, which will extend the existing evaluation from Chapter 5 to additionally validate our theory of system adaptation, including the proposed framing and our formal definitions. This could include introducing and comparing different Quality Functions—built according to the concrete system function that is considered to be adaptive, the concrete adaptation goals and uncertainties.

The previous research direction brings us to the next potential area of research in which researchers 1) can focus exclusively on conducting additional analysis on how the Quality Function is constructed and how it quantifies and qualifies the system function, and 2) also on how much the exposure under different partialities (previously discussed in Chapter 5) affect the calculation of the Quality Function. These analyses could be done on the created use case from the robotics domain, ideally extending to other application domains and systems in the future.

And finally, another big potential topic of research is implementing, evaluating, and testing different aspects of self-adaptive systems, especially the active self-adaptation, including the trigger for self-adaptation, and various other aspects and notions emerging from our formalisms.

System evolution. Finally, while discussing the second interpretation of the sequence of improving system functions in Chapter 5 and Appendix A, we explained that this could be considered as a sequence of *different functions* that evolve over time. For a function to change, it either changes its input or output interface (i. e., the domain spaces of the input and the output) or changes the state transition (i. e., the mapping of input to output within the same space domain of input and output) during run-time. We consider that in this case, we talk about system evolution, which is part of another research domain. Investigating the relation between system adaptation and system evolution is another potential future direction.

A A Theoretical Framework for Self-Adaptive Systems

In Chapter 5, we provided a framing and formal definition of *system adaptation*, discussed the preliminary characterization and minimum requirements for self-adaptive systems, and proposed a logical architecture for engineering self-adaptive CPSs that operate in dynamic and uncertain contexts. As part of this appendix, we extend the contributions and the formalization from Chapter 5 by proposing a theoretical framework for engineering self-adaptive systems. In our theoretical framework, we make the following contributions: (1) formally define passive and active self-adaptive systems using functions in which we embed the terminology that we proposed throughout this thesis, including its semantics, (2) we extend the minimum requirements for a system to be self-adaptive, and (3) we identify and specify the process of self-adaptation. In our formalization, we define both the structural (through the proposed terminology) and the behavioral aspects (through adopting functions for our formalization) of self-adaptive systems. Along with these contributions, we also (4) identify and discuss various emerging architectural implications. Namely, based on all our theoretical contributions, we propose a re-factored version of the MAPE-K conceptual model (for both passive and active self-adaptive systems) that adopts the terminology and is embedded within the proposed formalization. Lastly, (5) we exemplify our theoretical contributions on a well-known exemplar from the community.

In Appendix A.1, we first summarise the current state-of-the-art on the engineering of self-adaptive systems, including the limitations of the MAPE-K conceptual model for engineering self-adaptive systems and how self-adaptive systems are, in general, evaluated in the literature. The limitations of the MAPE-K conceptual model, to some degree, have been previously discussed in Chapters 1 and 5 and Section 8.2. In Appendix A.2, we summarize some of our prior theoretical concepts and findings from Chapter 5, which set the foundation for the formalizations and the contributions proposed in the rest of this appendix. In Appendix A.4, we formalize two types of self-adaptive systems: passive and active. In Appendix A.5, we exemplify our proposed theoretical framework on the Znn.com exemplar that was proposed by Cheng [27] in support of the Rainbow framework [44]. Finally, in Appendix A.6, we provide concluding remarks and discuss some limitations.

A.1 Current state-of-the-art and its limitations

As part of this section, we summarise the current state-of-the-art regarding the engineering of self-adaptive systems; concretely, we analyze: 1) the conceptual MAPE-K model, which is broadly accepted as a reference model for engineering self-adaptive systems (cf. Appendix A.1.1), and 2) the current best practices in the literature for evaluating self-adaptive systems (cf. Appendix A.1.2). Furthermore, we identify and discuss their limitations, which to some degree, served as a motivation for our work. We address their limitations with the contributions of this chapter.

A.1.1 Conceptual model of a self-adaptive system

According to the MAPE-K conceptual model, a self-adaptive system consists of a *managed element* and an *adaptation logic* (also known as managing element in the literature) [66]. The managed element is the entity that gains the ability to self-adapt as part of a self-adaptive system. It can be any system, e. g. software system or cyber-physical system (CPS). On the other hand, the adaptation logic is realized through the MAPE-K feedback loop. MAPE-K is an acronym for Monitor, Analyse, Plan, and Execute with shared Knowledge among all four phases. As previously explained in Section 1.1.2, the MAPE-K conceptual model was initially proposed in [66], and ever since, it has been used as a reference model for engineering not only self-adaptive systems but self-* systems in general.

This separation becomes additionally blurry with the introduction of different MAPE-based patterns for self-adaptive systems [123, 101] that focus on different combinations of decentralization of the four MAPE phases. Please note that these MAPE patterns do not explicitly consider the Knowledge component. Although different MAPE-based patterns could be more informative regarding the system's design, inherently, they have the same limitations as the MAPE-K closed feedback loop itself: 1) their high level of abstraction, without providing any characterization of how a system built upon the MAPE-K loop differentiates from the "ordinary," non-adaptive systems nor 2) a minimal set of requirements for the system to be considered as a self-adaptive.

This separation becomes additionally blurry with the introduction of different MAPE-based patterns for self-adaptive systems [123, 101] that focus on different combinations of decentralization of the four MAPE phases. Please note that these MAPE patterns do not explicitly consider the Knowledge component. Although different MAPE-based patterns could be more informative regarding the system's design, inherently, they have the same limitations as the MAPE-K closed feedback loop itself: their high level of abstraction, without providing any characterization of how a system built upon the MAPE-K loop differentiates from the "ordinary," non-adaptive systems nor a minimal set of requirements for the system to be considered as a self-adaptive.

A.1.2 Evaluating self-adaptive systems

The lack of clarity on what self-adaptive systems are (and their difference from the "ordinary," non-adaptive systems) correspondingly reflects on the aspects of their evaluation.

For this purpose, we have conducted a literature analysis in which we observed that the most common evaluation objectives in the existing works on self-adaptive systems (that include some evaluation) are not different from those generally used in the assessment of "ordinary," non-adaptive systems. For instance, effectiveness [59, 98], time efficiency (time for completion [97, 98], time for decision [88], response time [29]), scalability [110], robustness [59, 98], energy consumption [57, 99], resource cost [99], accuracy [59, 98], etc.

Our insights were additionally confirmed by a recent mapping study by Gerostathopoulos et al. [47], in which the authors analyze how self-adaptive systems have been evaluated in the SEAMS[1] community over the past decade. Their results showed that effectiveness,

[1]SEAMS stands for International Symposium on Software Engineering for Adaptive and Self-Managing

learning ability, time efficiency, scalability, and robustness were the evaluation objectives used in the experiments in the different papers that they analyzed.

In sum, none of the previous papers that include some evaluation of self-adaptive systems considers the *system adaptation* as an evaluation objective. This is primarily because the notion of *system adaptation* was not prior defined in the literature. A more precise understanding of *system adaptation* is not only necessary for defining self-adaptive systems but also for setting the foundation for a more systematic evaluation and even comparison of these systems in the future.

A.2 Summary of our previous theoretical findings and contributions on defining *system adaptation*

Defining what we understand under *system adaptation* is essential for a subsequent definition of self-adaptive systems, as we previously discussed in Chapter 4. To address this, in Chapter 5, we proposed a framing and a formal definition of system adaptation. In this dissertation, we concluded that in order to debate if a system is adaptive or not, we first need to specify the right framing according to which the system is considered adaptive:

[S] What is the **s**ystem function sf that is considered as adaptive?
[A] According to which **a**daptation goals does the system (i. e., the system function) adapt?
[C] According to which context or system **c**onditions (changes, uncertainties) does the system (i. e., the system function) adapt?

In the following, we explain these three aspects of our proposed framing in more detail. First, adaptation is a property of an individual system function sf. Instead of concentrating on the managed element (i. e., the system as a whole) as part of a self-adaptive system (see Appendix A.1.1), we shall focus on a concrete functionality of the system that gains adaptation capabilities. This system function fulfills the business goals of the system, i. e., it is related to the system's functional requirements. However, focusing on an individual function does not mean a simple function, and the "granularity" of the function matters.

Second, adaptation is intrinsically related to changing something, but always in order to improve certain adaptation goals. Contrary to the business goals, the adaptation goals are one or more quality objectives associated with specific non-functional requirements, as it can also be noted in various other works [114, 122, 119].

Finally, adaptation aims at preserving and improving the above-mentioned adaptation goals (i. e., quality objectives) when the system is exposed to internal and environmental (i. e., contextual) conditions that are changing and introduce uncertainties at run-time, which cannot be (fully) foreseen during the design of the system. Therefore, similarly to the system function from the first aspect of the framing, how a self-adaptive system is built depends on the concrete adaptation goals and the concrete context and system conditions (changes, uncertainties) according to which the system adapts.

Systems.

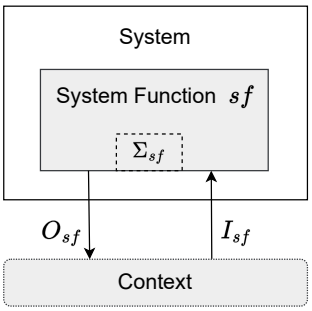


Figure A.1: *System function sf.*

Namely, the same system—precisely, system function sf—might be considered adaptive according to one adaptation goal and non-adaptive, according to another. Identically, the system cannot adapt to every condition: it might adapt to one set of conditions, whereas, according to another set of conditions, it could be considered non-adaptive. In sum, identifying the concrete system function, the adaptation goals, and the uncertain and changing conditions is paramount for discussing whether a system adapts or not; hence, it is essential for the specifications of self-adaptive systems.

As a second contribution in Chapter 5, we formally define *system adaptation* as follows. First, we assume that the behavior of a system at a given moment in time is determined by its system function sf (see Fig. A.1). The function sf maps the input of the system $i \in I_{sf}$, its internal state $\sigma \in \Sigma_{sf}$ and the context $\mathbb{C}$ to an output $o \in O_{sf}$ and new state $\sigma' \in \Sigma_{sf}$ (see Definition 1 in Chapter 5 [96]):

$$sf\colon I_{sf} \times \Sigma_{sf} \times \mathbb{C} \to O_{sf} \times \Sigma_{sf},\ f(i,\sigma) = (o,\sigma'), \tag{A.1}$$

where *context* ($\mathbb{C}$) is the part of the environment ($\mathbb{U}$) and consists of all the objects relevant to the system that affect the system's input (I_{sf}), and therefore, the state and behaviour of the system:

$$I_{sf} \subset \mathbb{C} \subset \mathbb{U}. \tag{A.2}$$

System adaptation is defined as a sequence of functions $sf_0, sf_1, \ldots, sf_m, \ldots, sf_n, \ldots$ at time $t = 0, 1, \ldots, m, \ldots, n, \ldots$, with $n > m$, that are assessed and their assessment improves over time (e. g., sf_n is "better" than sf_m) despite the changing and uncertain run-time conditions. The notion of "better" is determined by a newly introduced adaptation metric, to which the authors refer to as *Quality Function* Q. Defining Q, i. e., how the system function is assessed, what is "better" and how it is measured, is use case-specific and is left to the engineers of the self-adaptive system. However, the engineers need to consider and comply with the following rules in their definition and construction of the Quality Function:

1) Q needs to measure the degree of fulfillment of the business and the adaptation goals at a specific point in time. This includes considering and qualitatively assessing the context and the system state and quantifying their concrete realization. An example of how Q is constructed for a concrete system from the robotics domain is given in [96]. Independent of the concrete realization of the Quality Function, its value always resides within a fixed range, normalized between 0 and 1, as shown in Fig. A.2. The lower boundary is defined by the point where the business goals are no longer met, whereas the upper limit is reached when all of the business and adaptation goals are achieved perfectly.
2) Measuring Q at a single time instance (e. g., at time $t = 2$ in Fig. A.2) is not sufficient to reason how well the system adapts, or in general, if it adapts at all. Therefore, the

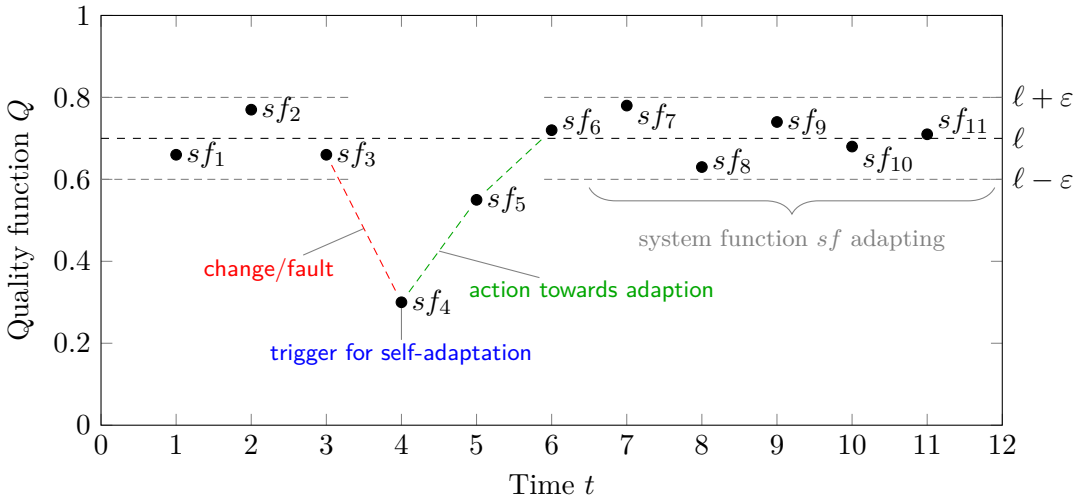


Figure A.2: *The process of self-adaptation, modified from [96].*

only way to evaluate whether a system adapts is to observe how Q develops over time.

3) Lastly, please note that a) Q cannot infinitely improve (there is only a limit to which it can improve), and b) Q does not necessarily asymptotically and monotonically improve over time. Instead, there are possible oscillations within a certain range after a certain time period (see sf_6 to sf_{11} in Fig. A.2). For that reason, we define that the sequence of Q should converge within a specific range after some time.

Definition 6 (Adaptive system [96])**.** An *adaptive system* is an infinite sequence of functions $(sf_i)_{i=0}^{\infty} = (sf_0, sf_1, \dots, sf_i, \dots)$ that satisfies some quality objectives, s.t. the Quality Functions Q of the infinite sequence of functions converges towards a limit ℓ s.t.:

$$\begin{aligned} \exists \ell, \varepsilon, i \; \forall t \geq i \colon |Q(sf_t) - \ell| < \varepsilon, \\ \text{with } \ell, \varepsilon \in [0,1] \text{ and } 0 \leq \ell - \varepsilon < \ell < \ell + \varepsilon \leq 1 \end{aligned} \tag{A.3}$$

where Q evaluates the quality of sf_t at time t, which converges after time i (see Fig. A.2 at $t = 6$). The values ℓ and ε are the convergence parameters defining the convergence threshold: ℓ is the convergence limit, and ε is the measure of the closeness of the convergence.

Figure A.2 illustrates the convergence of the Quality Function of a sequence of system functions sf_t with convergence limit $\ell = 0.7$ and measure of closeness $\varepsilon = 0.1$. Please note that in the figure, only a part of the infinite sequence of system functions is shown. Also, the Quality Function of the system functions series converges to the limit ℓ over time. Still, it does not necessarily converge to 1 (which would mean a perfect fulfillment of both business and adaptation goals that might not be feasible).

In response, we extend our framing with two more aspects:

[T] What is the **t**ime period in which the system is considered adaptive?
[C] Under which **c**onvergence parameters ℓ and ε is the system considered as adaptive?

Throughout the rest of the appendix, we will refer to our proposed framing as SACTC framing.

A.3 Exemplifying the SACTC framing using the robotics system

In the following we exemplify our proposed framing on the model problem from the robotics domain that is used throughout this dissertation.

[S] What is the **s**ystem function sf that is considered as adaptive?

→ *Adaptation is a property of a single function.*

[A] According to which **a**daptation goals does the system (i. e., the system function) adapt?

→ *The Quality Function Q is use case-specific and is constructed according to the specific adaptation and business goals.*

[C] According to which context or system **c**onditions (changes, uncertainties) does the system (i. e., the system function) adapt?

→ *To some uncertain conditions and situations, the system adapts and to some it does not, i. e., the range and the type of uncertainties and changes is important while specifying (self-)adaptive systems.*

[T] What is the **t**ime period in which the system is considered adaptive?

→ *Related to the concrete adaptation goals and uncertain conditions.*

[C] Under which **c**onvergence parameters ℓ and ε is the system considered as adaptive?

→ *Related to the concrete adaptation goals and uncertain conditions.*

Let us consider one or multiple mobile robots that autonomously traverse a room (i. e., their context) in order to clean it. The robots clean the room by discovering and attaining tasks (e. g., dirt patches) that continuously appear in the room with unknown time and location patterns (which are the external uncertainties). Autonomous, in this case, refers to their ability to navigate and move in the space without the need for external guidance, e. g., physical or electro-mechanical control [4, 75]. The robots discover the tasks with a LiDAR sensor, which has a smaller range of observation than the room size. In other words, the robots can only have a partial observation of the state of the room at a specific point in time $t = n$. Despite the partiality in the observations, the sensors introduce various other internal run-time uncertainties (e. g., sensor imprecision, ambiguity or failure). Finally, the robots attain or clean the tasks by driving to the tasks' locations.

In this use case, the robots provide thousands of different functionalities that can be considered potentially adaptive, and as discussed in the previous section, the "granularity" of the functions matters. For instance, we can focus on the Monte Carlo Localisation (a well-known particle filter algorithm [33]) that enables the robot to localize and navigate in the space, or we can focus on the function that queues the tasks for the robot, which defines the order in which the robot attains the tasks. Alternatively, we can also focus on the high-level functionality of the system: discovering the tasks and then moving to their locations to attain them. No matter which functionality of the system we focus on making adaptive, it is important to identify it explicitly. Although we focus on the same robotic system in the three cases, the objectives for building a self-adaptive system will

vary depending on which function we want to make adaptive. The first aspect of the framing also sets the foundation for the differentiation between system functioning and system adapting, which is the first step towards differentiating self-adaptive systems from the "ordinary," non-adaptive systems.

The robots fulfill their business goal by fulfilling the mission of the system, i. e., the managed element. However, in addition to keeping the room clean (which is the business goal in our use case), we also want to improve the quality of the cleaning process (the adaptation goal), despite the different run-time changes and uncertainties. How specific adaptation goals are defined and specified depends on the engineers of the self-adaptive system and their interpretation of the *quality of the cleaning process* for the concrete use case. For example, improving the quality of the cleaning process could mean cleaning as many tasks in the shortest amount of time or cleaning as many tasks while minimizing the distance that the robots traverse. Based on the specific interpretation and realization of the adaptation goals, the Quality Function is constructed. Please note that the same system (i. e., system function (sf)) might be considered adaptive according to one adaptation goal and maladaptive according to another.

As mentioned above, in the specific robotics example, there are 1) *external* run-time uncertainties, originating from the context in which the robots operate, concretely, the appearance of tasks for the robots with unknown time and location patterns, and 2) *internal* run-time uncertainties, originating from the system (i. e., the managed element) itself. For example, sensor failure, sensor imprecision, and sensor noise [104], as well as the partial sensing range of the robots. Both the external (the appearance of tasks at random locations) and the internal uncertainties (e. g., the sensor imprecision) lead to situations that affect the relevant aspects for the adaptation in this use case, i. e., the presence/absence of dirt tasks at a specific location in the room, which directly impacts the fulfillment of the adaptation goals (i. e., the system adaptation). As a simple example, consider that due to these run-time uncertainties, the robots end up driving to locations where there are no tasks (it still fulfills the business goal, but fulfillment the adaptation goal worsens—i. e., the value of Q drops). Additionally, while discussing system and context situations, it is important to emphasize that the system cannot adapt to every uncertain condition and situation (the range and the type of uncertainties are important to be specified), which is also in relation to the convergence parameters and the time period for which the system is considered adaptive. For example, if a hardware component of a system fails, then the system cannot even fulfill its business goal. In these cases, there might be no other alternative than terminating the system's operation until a human administrator intervenes and replaces the component (however, this directly opposes the core idea behind self-* systems, i. e., the increased system autonomy as discussed prior in the thesis). Similarly, the spawning rate of the tasks in the room might be so high that the robots—no matter how well they clean the space— are still not able to keep up with the quality of the cleaning process (i. e., the adaptation goal) due to the limited resources (e. g., the number of robots in the room, their velocity, etc.). Increasing the number of robots would be the necessary adaptation action in this particular case. We discussed this extensively through our evaluation in Chapter 5.

A.4 Defining Self-Adaptive Systems

A.4.1 Two premises in defining self-adaptive systems

The term self-adaptive software was used for the first time in the literature by Robert Laddaga from MIT in DAPRA technical report [77] in 1997. The informal definition of self-adaptive software that Laddaga has given as part of this technical report could be considered, to this day, as superior and more complete compared to all the other informal definitions that have been published in the literature ever since. Concretely, according to Laddaga's definition [77]:

1) "we seek a new basis for making software adaptive, that does not require specific adaptive techniques, such as neural networks or genetic programming, but instead relies on software informed about its mission and about its construction and behavior," and
2) "self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible."

Based on the first part of his definition, we can argue that *the adaptation logic as a component of the self-adaptive system already enables the system to be informed about its mission, construction, and behavior*. However, as extensively discussed throughout this dissertation, if the MAPE-K closed feedback loop is exclusively used as a reference model for building the self-adaptive system, then very misleadingly, every system that incorporates some aspects of monitoring, analysis, planning and execution can be labeled as a self-adaptive system. In response, in many of the theoretical contributions in this dissertation: 1) the SACTC framing, 2) the necessity to consider and separate the adaptation and the business goals and the implications of the separation of the goals on the design of self-adaptive systems, as well as 3) the preliminary characterization of self-adaptive systems, we focused on improving the semantic clarity of the adaptation logic. Our findings supported and also complemented the ideas that Laddaga positioned in the first part of his informal definition.

However, the second part of Laddaga's definition is almost entirely overlooked in the existing approaches for engineering self-adaptive systems that have been proposed in the literature, as well as all the other informal and formal definitions of self-adaptive systems. In our view, this is because the Vision of autonomic computing by Kephart and Chess [66], which proposed the MAPE-K closed feedback loop and became the de facto model for engineering self-adaptive (and self-* systems), was published a few years after Laddaga's definition and completely excluded the ideas that Laddaga positioned on self-adaptive systems evaluating themselves regarding the fulfillment of their goals.

With the contributions in this appendix, building upon the definition of system adaptation and the SACTC framing from Chapter 5, we close this gap. Precisely, in this thesis, to measure system adaptation, we proposed a metric to which we referred as *Quality Function Q*, which quantifies the fulfillment of the adaptation goals (also the business goals) and serves as a foundation for the realization of the second part of Laddaga's definition. The Quality Function: 1) inherently enables the system to evaluate its own behavior and

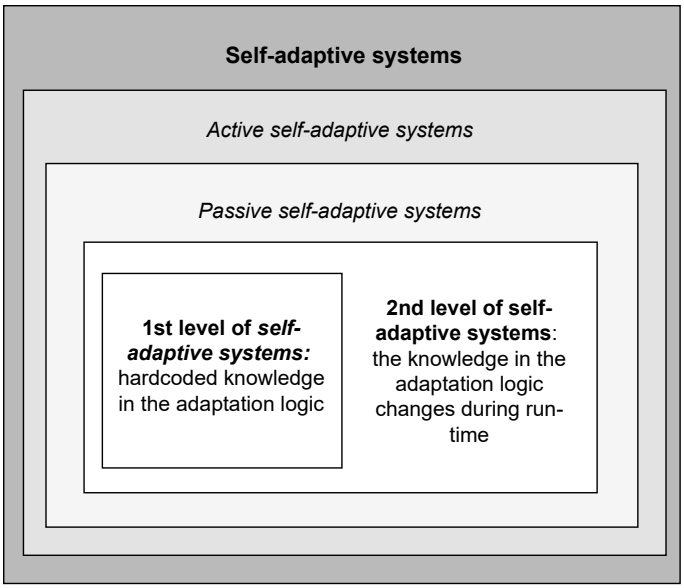


Figure A.3: *Different levels and types of self-adaptive systems and their relations.*

2) indicates when the system is not accomplishing what it is intended to do, i. e. when the adaptation goals are not fulfilled within a certain range (which then becomes the trigger for self-adaptation, see Fig. A.2). In response, the self-adaptive system should be able to choose different actions towards adaptation (see Fig. A.2) for the system to (eventually) adapt. Eventually means that the system adaptation (i. e., the satisfaction of the adaptation and the business goals within a certain range) will happen either immediately in the next time step (e. g., with a single action towards adaptation) or might happen later (e. g., after multiple adaptation actions).

A.4.2 Classification of self-adaptive systems

In order to be inclusive of all the existing works in the literature that do not consider the second part of Laddaga's definition, we differentiate between two types of self-adaptive systems: active and passive self-adaptive systems (see Fig. A.3). We make this differentiation based if the value of the Quality Function Q is used passively or actively (as a self-adaptation trigger) in the self-adaptation:

1) *passive self-adaptive systems*, depicted in Fig. A.4a, in which the system (i. e., the managed element) is informed about its mission, construction and behavior through the existence of the adaptation logic (as envisioned by Laddaga in the first part of his definition), and
2) *active self-adaptive systems*, depicted in Fig. A.4b, in which besides the existence of the adaptation logic the system can evaluate itself and can steer the adaptation based on the value of the Quality Function (comprising the first and the second part of Laddaga's definition).

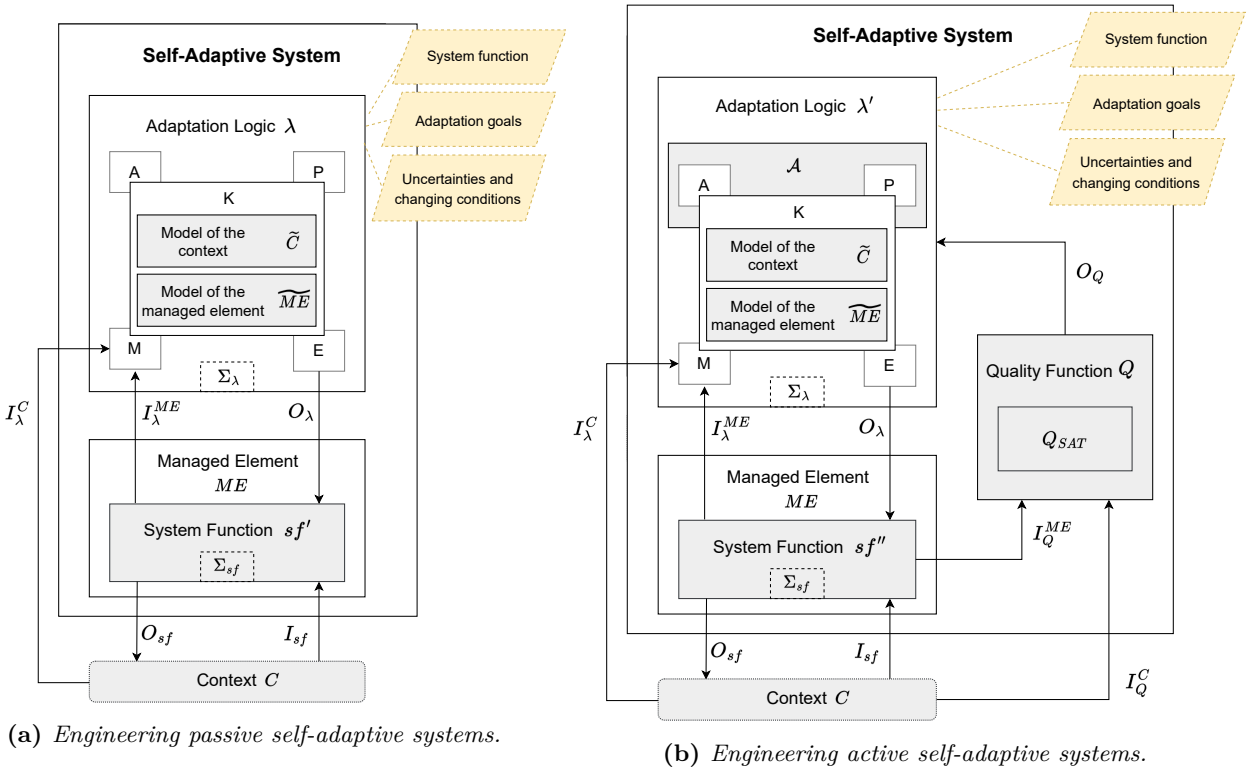
Please note that in both of the cases, regardless if we consider passive or active self-

adaptation *the value of the Quality Function needs to be measured*, in order to discuss if a system adapts or not. As previously defined in Chapter 5 and summarized in Appendix A.2, discussing if a system adapts or not is impossible 1) without the respective SACTC framing and 2) without measuring the Quality Function and checking if it adheres to the concrete rules specified in its definition. The only difference is that in passive self-adaptation, we assume that if the adaptation logic is built according to the framing and the concrete specifications that we propose in this thesis, then it will, by default, improve the state of the system (i. e., the managed element), which in result will improve the value of Q. Whereas, in active self-adaptation, besides the existence of the adaptation logic, the measured value of the Quality Function Q is used by the self-adaptive system. Namely, when Q is out of the convergence range, then it becomes a self-adaptation trigger (see Fig. A.2). We explain this further in the following.

To summarize, in the passive self-adaptive systems, the adaptation logic is constructed according to the concrete system function, the adaptation goals, and the uncertain conditions (depicted with yellow parallelograms Fig. A.4a). As said above, it is expected that the adaptation logic improves the state of the system (i. e., the system function (sf)) according to the concrete adaptation (and business) goals, which as a result, will improve the value of Q. This type of self-adaptation can be considered as a more indirect way of self-adaptation since it happens *implicitly* through the existence of adaptation logic without an active trigger for self-adaptation. It is important to point out that although various theoretical contributions from this dissertation improved the clarity of the engineering processes for self-adaptive systems, including the engineering of passive self-adaptive systems, the overall self-adaptation process still remains underspecified for this type of self-adaptive systems, and it could be a source of confusion and misunderstanding.

This is addressed in the active self-adaptive systems through the "active" usage of the value of the Quality Function. Concretely, in the active self-adaptation, besides the existence of the adaptation logic, the measured value of the Quality Function is also actively used as input to the adaptation logic. When Q indicates that the system does not do what it is intended to do (i. e., the value of Q drops below the range of convergence), then in response, a concrete adaptation action from the adaptation logic is chosen. This enables the system to evaluate itself and to indicate when it is not accomplishing what it is intended to do. The idea of the measured Q actively steering the self-adaptation (through the trigger for self-adaptation, see Fig. A.2) incorporates both of the aspects from the informal definition of self-adaptive software by Laddaga and can be considered as a "feature-complete" or true self-adaptation. As part of the active self-adaptive systems, the need emerges for additional mechanisms in the self-adaptive system, which use the value of the Quality Function (i. e., the trigger for self-adaptation) to choose the best actions and strategies towards adaptation.

Additionally, as seen in Fig. A.3 we differentiate between two different levels of self-adaptive systems: the first and the second level of self-adaptive systems. We make this distinction depending if the knowledge in the adaptation logic is hard-coded during the design of the self-adaptive systems (first level of self-adaptive systems), or if the knowledge changes and gets updated (i. e., "is learns") during the run-time operation of the self-adaptive system (cf. Fig. A.3). It is important to point out that the first and the second



(a) *Engineering passive self-adaptive systems.*

(b) *Engineering active self-adaptive systems.*

Figure A.4: *Our re-factored version of the MAPE-K conceptual model.*

level of self-adaptive systems are orthogonal to passive and active self-adaptive systems. Therefore, we can talk about the first level of passive or active self-adaptive systems and the second level of passive or active self-adaptive systems. Additionally, the first level of self-adaptive systems corresponds to the intermittent autonomy from Chapter 3, and the second level of self-adaptive systems corresponds to both the eventual autonomous and to the fully autonomous systems, depending on the sophistication of the learning capabilities.

In Fig. A.4, we show our re-factored version of the MAPE-K conceptual model for engineering self-adaptive systems for both passive and active self-adaptive systems. In these re-factored MAPE-K conceptual models, we only distinguish between passive and active self-adaptive systems and not according to the first and second level. The components depicted in grey in the re-factored versions of the MAPE-K conceptual model are the new components that we propose as part of our work. We use our re-factored versions of the MAPE-K for the needs of the formalisation of passive and active self-adaptive systems that we propose in the following sections.

A.4.3 Defining (first and second level of) passive self-adaptive systems

Previously in this chapter, as well as in the characterization of self-adaptive systems in Chapter 5, we concluded that the adaptation logic for a concrete system (i. e., managed element) needs to be built according to the concrete system function, the adaptation goals, and the uncertain conditions according to which the system adapts (see Fig. A.4). If we

assume that 1) the adaptation logic is implemented based on the MAPE-K closed feedback loop and that 2) the implementation of the four MAPE phases (monitor, analyze, plan, and execute) can be distributed from the adaptation logic to the managed element (i. e., the system) as various patterns in the literature propose [123, 101]; then the minimum requirements for the adaptation logic in passive self-adaptive systems is the existence of knowledge component K created correspondingly to the SACTC framing.

The knowledge K in the adaptation logic should consist of information about the system (i. e., the managed element) and the context that is relevant for the concrete adaptation. Concretely the knowledge in the adaptation logic should reflect the state or the behavior of: 1) the system (i. e., the managed element), 2) the context, or 3) both, by keeping models of their relevant aspects for the concrete adaptation in K.

There could be various ways to realize the adaptation logic, including the knowledge, which is independent of our definition. How the knowledge is represented and constructed is something that we do not prescribe as part of our theoretical framework, nor our logical architecture from Chapter 5. However, please note that this changes in the contributions from Chapters 6 and 7, where we model the knowledge in the adaptation logic as a grid map. Also, the knowledge can be hard-coded in the adaptation logic during the design of the self-adaptive system (in the first level of self-adaptive systems), or *learned* and modified during the operation of the self-adaptive system to continue reflecting the actual, real-time state of the relevant aspects from the system (i. e., managed element) and the dynamic context (in the second level of self-adaptive systems). The latter introduces various new research directions that we tackle as part of Chapters 5 to 7. For example, run-time reasoning based on which the knowledge is updated, reasoning under uncertain observations, etc.

Hard-coded knowledge in the adaptation logic will likely satisfy the quality objectives from the adaptation goal(s) if all the run-time states of the context and the managed element can be anticipated during the system's design and they do not change beyond that. However, this is not feasible for every system, e. g. for dynamic systems like CPSs, where the observable behavior, state, or structure of the context and the system (i. e., the managed element) change in a way that they cannot be (fully) anticipated and known during the design. To exemplify, we refer to our model problem from the robotics domain. In our model problem, due to the run-time uncertainties from the context (i. e., the room) in which the robots operate, it is impossible for the designers of the system to anticipate the exact spatial patterns of the tasks' appearance. Therefore, if the knowledge of the adaptation logic is hard-coded during the design of the self-adaptive systems, it will encode only an approximation of the tasks' appearance known during the design, although the patterns of tasks' appearance might be completely different during the system's operation. As a result, instead of hard-coded knowledge in the adaptation logic of self-adaptive systems, the knowledge (i. e., the models in the knowledge) should get modified and updated during run-time, i. e. *learned*, to reflect the run-time state of the relevant aspects from the system (i. e., the managed element) and the context for the concrete adaptation. Often in the literature, these models are referred to as models at run-time (models@RT), which are considered as a support for realizing self-adaptive systems [13, 12, 115]. Please note that we do not consider state machines as run-time models.

To update the knowledge at run-time, the adaptation logic requires some mechanisms for reasoning and storing the newly obtained information, which comprises the *learning* aspect, based on which the adaptation logic changes during the run-time. This enables the self-adaptive system, concretely the knowledge in the adaptation logic, to continue reflecting the actual state (i. e., the state during the operation of the self-adaptive system) of the context and the system (i. e., the managed element). In response, in the adaptation logic, we differentiate between two types of knowledge: 1) knowledge at design (K_D) that is hard-coded by the designers in the adaptation logic at design time, and 2) learned knowledge (K_L), which is updated by the self-adaptive system during its run-time, based on which we differentiate two types of passive self-adaptive systems:

1) first-level passive self-adaptive systems with hard-coded knowledge in the adaptation logic, and
2) second-level passive self-adaptive systems in which the knowledge in the adaptation logic changes and improves over time.

Formalising passive self-adaptive systems

In this thesis, we adopt an approach to formally define self-adaptive systems using functions. Namely, in passive self-adaptive systems, the function of a self-adaptive system f_{SAS} is a composite function of λ (the function of the adaptation logic) and sf' (the system function that gains adaptation capabilities as part of the managed element), as it also can be seen in Fig. A.4a:

$$f_{SAS} = \lambda \circ sf'. \tag{A.4}$$

In Equation (A.1) in Appendix A.2 (as well as in Definition 1 in Chapter 5), we defined the system function sf as the mapping of the system's input I_{sf}, its internal state Σ_{sf} and the context $\mathbb{C}$ to an output O_{sf} and new state Σ_{sf} (see Fig. A.1):

$$sf\colon I_{sf} \times \Sigma_{sf} \times \mathbb{C} \to O_{sf} \times \Sigma_{sf},$$

Please note that during the system's operation, there are parts of the context $\mathbb{C}$ that the system (i. e., system function) can perceive and determine through its input interface I_{sf}, and parts of the context that the system cannot determine through its input—e. g., in CPSs due to missing sensors—but still have effects on the system and its behavior. We refer to the encoded input with explicit interfaces I_{sf} as direct inputs (cf. I_D in Fig. A.5) and to parts of the context for which there is not a direct input but still have an effect on the system, but we refer to as indirect inputs (cf. I_I in Fig. A.5). Since $I_I \subset \mathbb{C}$, we included only $\mathbb{C}$ as part of Equation (A.1).

Since we define a function by the mapping between its input and state to an output and an updated state, for that reason, as part of a self-adaptive system (as shown in Fig. A.4a) instead of sf we refer to sf' which has an additional pair of input and output to the adaptation logic, compared to sf from Eq. (A.1) (as shown in Fig. A.1). Concretely, we define sf' as following:

$$sf'\colon I_{sf} \times O_{\lambda} \times \Sigma_{sf} \times \mathbb{C} \to O_{sf} \times I_{\lambda}^{ME} \times \Sigma_{sf}, \tag{A.5}$$

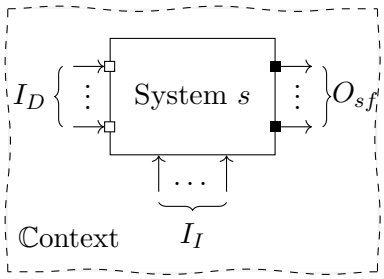


Figure A.5: *Direct I_D and indirect I_I inputs to the system.*

where is O_λ is the output from the adaptation logic to the managed element and I_λ^{ME} is the input from the managed element to the adaptation logic.

To again summarize, if we compare sf in Equation (A.1) and Fig. A.1, with sf' in Equation (A.5) and Fig. A.4a, the fact that there are additional input (O_λ) and outputs (I_λ^{ME}) in sf' when the managed element communicates with the adaptation logic, differentiates the system function sf' from sf when the managed element is stand-alone. However, please be aware that we did this differentiation only for mathematical precision of our formalism, and the system functions sf and sf' have a behavioral equivalence, e. g. from a user standpoint if the system is considered as a black box.

In the **first level of passive self-adaptive systems** (as shown in Fig. A.4a), we define the function of the the adaptation logic λ^1 as follows:

$$\lambda^1 \colon I_\lambda \times \Sigma_\lambda \to O_\lambda \times \Sigma_\lambda,$$

where Σ_λ is the state of λ^1, and the knowledge $K_D = \widetilde{ME} \times \widetilde{C}$ is a subset of Σ_λ ($K_D \subset \Sigma_\lambda$. $\widetilde{ME}$ is the model of the system and $\widetilde{C}$ is the model of the context in the knowledge of the adaptation logic. As previously discussed in Chapter 5), the models of the managed element $\widetilde{ME}$ and the context $\widetilde{C}$ are abstractions or approximations of the relevant aspect for the adaptation from the concrete managed element ME and context $\mathbb{C}$. Lastly, the input space of I_λ is a Cartesian Product of the input from the managed element I_λ^{ME} and the input from the context I_λ^C ($I_\lambda = I_\lambda^{ME} \times I_\lambda^C$).

Depending on the architecture of the self-adaptive system and the exact system (i. e., managed element) under consideration, the input from the context to the adaptation logic might not exist. In that case, I_λ^C is an empty set. As a result, all the information that the adaptation logic (i. e., the adaptation logic function λ) has from the context is propagated through the inputs I_{sf} from the context to the managed element (i. e., the system function) and I_λ^{ME} from the managed element to the adaptation logic. This introduces some research challenges and technical problems on its own, e. g., how to aggregate different (partial) observations from the context that multiple systems make independently. We identify some of those challenges in our proposed logical architecture for engineering self-adaptive CPSs, in Chapter 5, and address them in Chapters 6 and 7 of this thesis.

If the adaptation logic is engineered according to the MAPE-K feedback loop, then the adaptation logic function λ is a composite function of the functions m, a, p, e of the phases

M, A, P, E:

$$\lambda = m(\ldots, k) \circ a(\ldots, k) \circ p(\ldots, k) \circ e(\ldots, k),$$

where $k \in K$ is the shared knowledge among all the MAPE phases. As extensively discussed throughout the paper, the construction of the adaptation logic is always dependent on the concrete system function, the adaptation goals and the changing and uncertain conditions according to which the system adapts, as shown in Fig. A.4a.

At the first level of passive self-adaptive systems, the knowledge in the adaptation logic does not change beyond what was specified by the engineers during the system design and development of the self-adaptive system. Therefore, the improvements that the adaptation logic on the first level is offering are limited. Namely, hard-coded adaptation logic does not suffice when the managed elements are dynamic systems, e. g., CPSs that are exposed to various run-time uncertainties. In sum, an adaptation logic whose knowledge does not enhance during run-time is not sufficient to continuously optimally fulfill a given quality objective, especially when modern dynamic systems, e. g. CPSs are in focus. These systems operate in changing, and uncertain run-time situations that need to be considered and represented in the knowledge of the adaptation logic but could not be known during the design phase of the system. Also, please note that the functions m, a, p, e from the MAPE phases could incorporate algorithms that utilize some different (machine) learning techniques, which do not impact or change the knowledge component in the adaptation logic. We distinguish between the first and the second level of passive self-adaptation only in terms of whether the knowledge in the adaptation logic changes or not. However, one could also argue that between the current first and second level of self-adaptation, multiple other orthogonal levels can be introduced based on different algorithms' sophistication for the other MAPE phases.

In the **second level of passive self-adaptive systems** (cf. Fig. A.3), the adaptation logic is a function λ^2 is defined as following:

$$\lambda^2 \colon I_\lambda \times \Sigma_\lambda \times K_L \to O_\lambda \times \Sigma_\lambda.$$

On this level, K_L is learned by the *lrn* that is a recursive function that aggregates observations and updates the knowledge (i. e., the models of the managed element $\widetilde{ME}$ and the context $\widetilde{C}$) in the knowledge of the adaptation logic during run-time:

$$lrn \colon K_L \times I_\lambda^{ME} \times I_\lambda^C \to K_L, \quad lrn(k_L^n, i_\lambda^{ME}, i_\lambda^C) = k_L^{n+1}, \text{ where } k_L^0 = k_D.$$

The adaptation logic in the second level of passive self-adaptive systems is engineered in a way that it has the ability to change and enhance the knowledge that has been initially specified during the system's design (K_D), based on the new observations gathered, aggregated and analyzed, based on which the knowledge is updated during run-time.

Limitation of passive self-adaptive systems

As previously discussed in Appendix A.4.2, the overall self-adaptation process still remains underspecified for this type of self-adaptive system, and it could be a source of confusion and misunderstanding. According to our definition of system adaptation in Chapter 5, we can clearly identify when a system adapts; however, the exact process of self-adaptation in passive self-adaptive systems remains unclear and fuzzy. We tackle this issue in the definition of active self-adaptive systems, and as a foundation for it, we exactly use the definition of system adaptation.

A.4.4 Defining active self-adaptive systems

In the passive self-adaptation, we assume that the existence of the adaptation logic improves the state of the managed element (i. e., the system) that results in an improved value of the Quality Function Q, compared with the case when the system is operating without the input from the adaptation logic. In this case, we say that the value of Q is used passively since we only use it to measure the system adaptation. If the value of Q is used "actively" 1) to detect when the system adaptation is not fulfilled (e. g., through the trigger for self-adaptation, see Fig. A.2), 2) based on which different actions towards adaptation can be chosen for the adaptation to be eventually achieved again, then we say that the value of Q is actively used by the self-adaptive system. Please note that the active self-adaptive systems are an extension of the passive self-adaptive systems, and at the same time, they are orthogonal to the first- and the second-level of self-adaptive systems (i. e., the knowledge in the active self-adaptive system can be both hard-coded in the adaptation logic, or it could improve during run-time).

The process of self-adaptation

As shown in Fig. A.2, the process of *self-adaptation* contains the following three steps (or four steps, if the zeroth step is considered):

Step 0: Drop in the value of Q (depicted with red in Fig. A.2);
Step 1: Trigger for self-adaptation (depicted with blue);
Step 2: Action(s) towards adaptation (depicted with green);
Step 3: Fulfilled system adaptation (depicted with grey).

Only when these steps are considered, then we can proceed with the specification of the self-adaptive system. We explain all the steps in more detail in the following.

The value of the Quality Function Q might drop due to various changing and uncertain conditions from the system (i. e., the managed element), the context, or both. The concrete reason for the drop of Q is application or system-dependent, and it needs to be considered in the concrete specifications or the design of the self-adaptive systems. Also, the system cannot adapt to unknown unknowns, and the range and the type of uncertainties and changes are important to be identified while specifying self-adaptive systems [85].

Previously, in Chapter 5 and Appendix A.2, we introduced the Quality Function as a metric for quantifying and measuring the system adaptation. Concerning the Quality

Function, in this section, we introduce a *Quality Satisfaction Function*, Q_{SAT}, based on which the need for self-adaptation is triggered, and adaptation actions are initiated. The Quality Satisfaction Function is a Boolean function, which as parameters in its signature, takes the value of the Quality Function (a real number between 0 and 1) and the convergence parameters ℓ and ε (cf. Definition 1 in Chapter 5):

$$Q_{SAT}\colon Q \times \ell \times \varepsilon \rightarrow \{\mathsf{true}, \mathsf{false}\}. \tag{A.6}$$

The trigger for self-adaptation (see Fig. A.2) is when the Quality Satisfaction Function becomes false. Concretely, this happens when the value of the Quality Function Q, or the fulfilment of adaptation (and the business) goals, is outside the range specified by the convergence parameters. In response, adaptation actions are initiated by the adaptation logic with *an intention for the system to adapt eventually* (i. e., Quality Satisfaction Function Q_{SAT} to become true). Eventually means that the adaptation could happen immediately after a single adaptation action or after a sequence of multiple adaptation actions. To summarize, the Quality Function and the Quality Satisfaction Function set the foundation for the self-adaptive system to evaluate itself and indicate when it is not accomplishing what it is intended to do, as envisioned by Laddaga. Based on this, the adaptation logic that reflects the structure and the behaviour of the managed element and the context chooses the best adaptation action at a specific point in time from the set of possible adaptation actions (see $\mathcal{A}$ in Fig. A.4b). The adaptation action is passed to the managed element, or more precisely, the system function sf. As a result of all of the above, as part of this paper, we extend the SACTC framing proposed in [96] with one more dimension that is necessary to specify self-adaptation (becoming SACTCA framing):

[A] What are the adaptation **a**ctions (or the actions towards adaptation) that the adaptation logic can pass to the managed element after an active trigger for self-adaptation?

Exemplifying the SACTCA framing using the robotics system cont. Referring again to the exemplification of the framing in Appendix A.3, we need to specify what actions towards adaptation can be undertaken if the tasks for the robots are spawned with such a high rate that the robots cannot keep up with their cleaning (reflected in the reduction of the Quality Function as shown in the evaluation in Chapter 5). A potential adaptation action could be including more robots in the collaboration, or prioritizing the cleaning of the tasks according to some rules, or having mechanisms which allow the robots to communicate with each other and share information about areas of the room that get dirtier than others. It is important to emphasize that this depends on the concrete adaptation goals and the system's overall functionality. Also, as we previously defined before, a system self-adapts with the intention for adaptation to eventually be achieved. We also said that eventually means that the adaptation could happen immediately after a single action towards adaptation, or multiple of them. Therefore, different actions toward adaptation will have a different trend toward achieving system adaptation. The extended SACTCA framing is the complete framing that needs to be considered during the design of self-adaptive systems.

In a nutshell, while specifying active self-adaptive systems, the system's designers and engineers also need to specify the concrete adaptation actions available that will enable

the system's self-adaptation. We detail further the presented self-adaptation process and the specifications in the following sections.

Extension of the minimum requirements for active self-adaptive systems

In Appendix A.4.3, we discussed that for passive self-adaptive systems, the minimum requirements for the adaptation logic is the existence of knowledge component K created correspondingly to the SACTC framing. However, in active self-adaptive system, there is one more requirement for the adaptation logic, and that is having a set of all the possible adaptation actions $\mathcal{A}$. Similarly as the knowledge in the adaptation logic, the set of the adaptation actions can also be hard-coded in the adaptation logic during the design of the self-adaptive system, parametrized, or completely learned during the run-time of the self-adaptive system. As a result, this introduces various new research directions in this domain. It is important to point out that integrating this last part of the theoretical framework in relation to the adaptation actions remained out of the scope of our logical architecture, and all the technical contributions of this thesis. Namely, our logical architecture, integrates the Quality Function and the Quality Satisfaction Function as Quality Evaluator (see Chapter 5), however, we did not put any emphasis on the consideration and the importance of the set of the possible adaptation actions.

If we only consider the adaptation logic as part of the self-adaptive system, then we only cover the first part of Ladagga's definition, as we previously explained at the beginning of Appendix A.4.1. Not to exclude the majority of the existing works in the literature, that engineer their self-adaptive (and in general self-*) systems according to the MAPE-K, we referred to these self-adaptive systems as passive self-adaptive systems. As explained at the beginning of this section, both aspects from Laddaga's informal definition of self-adaptive software are considered if the adaptation logic (to whose semantic clarity we have contributed in this dissertation) is additionally accompanied by the usage of the value of the Quality Function Q. This also enables us to pinpoint the process of self-adaptation clearly and precisely.

Above we summarized the minimum requirements for the adaptation logic in active self-adaptive systems. In the following, we summarize the minimum requirements for active self-adaptive systems that have various architectural implications for engineering these systems (see Fig. A.4b).

1) identification of the concrete system function from the managed element that gains adaptation capabilities as part of the self-adaptive system, the concrete adaptation goals and the concrete uncertain conditions,
2) existence of the adaptation logic built according to the concrete system function, the adaptation goals and the uncertain conditions (depicted with yellow parallelograms Fig. A.4b), whose knowledge keeps models of the relevant aspects for the adaptation from the managed element and the context,
3) definition of the Quality Function that enables quantification and measurement of system adaptation and serves as a foundation for the self-adaptive system to evaluate itself, based on which the self-adaptation is triggered when the adaptation and the

business goals are not fulfilled within a certain range, and lastly, once the self-adaptation is triggered, and

4) consideration and identification of the set of the possible adaptation actions $\mathcal{A}$.

Formalizing active self-adaptive systems

The function of an active self-adaptive system f_{SAS} is a composite function of λ', sf'' and Q, as it also can be seen in Fig. A.4b:

$$f_{SAS} = \lambda' \circ sf'' \circ Q. \tag{A.7}$$

Similarly as in Equation (A.1) in Appendix A.2 and Equation (A.8) in Appendix A.4.3, we define sf'' as following:

$$sf'' \colon I_{sf} \times O_\lambda \times \Sigma_{sf} \times \mathbb{C} \to O_{sf} \times I_\lambda^{ME} \times I_Q^{ME} \times \Sigma_{sf}, \tag{A.8}$$

where O_λ is the output from the adaptation logic to the system function and contains a specific action towards adaptation a ($a \in O_\lambda \subseteq \mathcal{A}$) at a specific point in time. Let I_λ^{ME} be the input from the managed element (i. e., sf'') to the adaptation logic, and I_Q^{ME} be the input from the managed element (i. e., sf'') to the quality function.

As previously explained, the fact that there are additional input (O_λ) and outputs (I_λ^{ME} and I_Q^{ME}) in sf'' when the managed element communicates with the adaptation logic and the quality function, differentiates the system function sf'' from sf' in the passive self-adaptive system and from sf when the managed element is stand-alone. However, all the three functions have a behavioural equivalence. This holds as well for the adaptation logic (cf. λ in Fig. A.4a and λ' in Fig. A.4b).

Concretely, in active self-adaptive systems we define the function of the adaptation logic λ' as following:

$$\lambda' \colon I_\lambda^C \times I_\lambda^{ME} \times O_Q \times \Sigma_\lambda \to O_\lambda \times \Sigma_\lambda,$$

where O_Q is the output from the Quality Function passed as an input to the adaptation logic. The output O_Q passes the trigger for self-adaptation to the adaptation logic when $Q_{SAT} = \mathsf{false}$, as previously defined in Equation (A.6). Figure A.4b also shows I_Q^{ME} and I_Q^C, which are the inputs that the Quality Function receives from the managed element and the context. In Appendix A.2 and Chapter 5, we discussed that the notion of "better" and how system adaptation is measured is determined by the quality function Q. However, the concrete definition and specification of Q are use case-specific, and Q can only be constructed by the engineers of a self-adaptive system. The only prerequisite for the engineers is to comply with the rules for the construction of the Quality Function that we summarized in Appendix A.2. Regardless of the a concrete approach or technique for constructing the Quality Function, all the potentially adopted solutions for Q in the future will need to consider some information from the system function sf'' and the context, which will be received through the inputs from the managed element I_Q^{ME} and the context I_Q^C that we depict in Fig. A.4b. We proposed one approach to construct Q for the concrete robotic system as a separate contribution in Chapter 5.

A.5 Exemplary Use Case

To demonstrate the theoretical contributions on engineering active self-adaptive systems, we choose the popular Znn.com[2] exemplar from the literature. In this section, we only descriptively exemplify how the proposed theoretical framework applies to Znn.com. Further empirical evaluation remained out of the scope of the current work.

Znn.com was proposed by Cheng [27] in support of the Rainbow framework [44]. Please note that the terminology we propose and use as part of this thesis, including its semantics, does not exist in the Rainbow framework. With the contributions in this paper, we make *explicit* different notions that were only intuitively and implicitly used by the authors of the Rainbow framework and Znn.com. It is also important to note that we deliberately chose Znn.com to exemplify our theory because for many of the other available examples in the field, we could neither find arguments nor an underlying intuition as to why the authors consider their systems to be self-adaptive (and how the nominal system function differs from self-adaptation in their use cases). This made it impossible for us to map other examples onto the proposed theoretical concepts. However, these insights served as additional validation for the importance of our contributions and why improved terminological clarity and specifications of these systems are of crucial importance.

Znn.com is a web-based client-server system and a news service that serves multimedia news content to its customers, which is the *business goal* of Znn.com. Using a load balancer, the system balances requests across a pool of replicated servers. Balancing the requests is the concrete *system function* that gains adaptation capabilities in the exemplar. As can be noted, this is one specific function out of the many functionalities that Znn.com has as a web-based client-server system and a news service. From time to time, Znn.com experiences drastic increases in news requests that are not within the range of the originally designed parameters (i. e., the requests exceed the capabilities of the servers required for normal workloads), resulting in the clients receiving their content with time delays. During the system's design, it is unknown when the peak periods of requests will occur, how high the peaks are and how long they will last, which are the *changing and uncertain conditions* in the system. The spikes in client requests cause a performance drop and cost increase. The performance and the cost are the concrete quality objectives or *the adaptation goals* in Znn.com, and the system adapts to improve these adaptation goals despite the changing and uncertain conditions. Concretely, in Znn.com, when the concrete adaptation goals *are not fulfilled* (i. e., $Q_{SAT} = \mathsf{false}$, which activates the trigger for self-adaptation), then 1) the server pool size is adjusted, or 2) the content is switched between multimedia and textual modes, which are the *actions towards adaptation* in this concrete system. Namely, the authors propose four possible actions towards adaptation:

- a_1 Switch the server content mode from multimedia to textual,
- a_2 Switch the server content mode from textual to multimedia,
- a_3 Increment the server pool size, and
- a_4 Decrement the server pool size.

[2] https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/model-problem-znn-com/

Different actions towards adaptation will enable the adaptation goals to be eventually fulfilled again (i. e., $Q_{SAT} = \mathsf{true}$ within a certain range of values).

In Znn.com, the authors do not explicitly identify and frame the system function that gains adaptation capabilities. However, we can identify that they still implicitly distinguish the concrete system function sf that gains adaptation capabilities from the other functions. Also, it can be noted that the authors implicitly separate the concerns of the system function and the function of the adaptation logic. Furthermore, we can observe that in their exemplar, the adaptation logic is also built based 1) on the concrete uncertainties, which in Znn.com are the spikes in the requests, and 2) the concrete adaptation goals: the system performance and the cost, for which in Znn.com the authors searched for a trade-off in their fulfilment. We can argue that if, in Znn.com, a different system function, adaptation goals, and uncertain conditions were under consideration, then the adaptation logic would have needed to be built differently, therefore, providing a completely different adaptation function λ. As discussed throughout the paper, considering these three aspects in the specifications of the self-adaptive systems is crucial for the later engineering of these systems.

Additionally, the general idea of:

1) the newly introduced concept of a Quality Function Q that measures the system adaptation (i. e., that quantifies and qualifies the fulfilment of the adaptation and business goals),
2) improving the system function sf over time according to a concrete assessment (i. e., the quality function),
3) the trigger for self-adaptation when there is a drop in the quality function (i. e., when the quality satisfaction function Q_{SAT} becomes false), and
4) the actions towards adaptation that enable adaptation to be eventually fulfilled again (i. e., $Q_{SAT} = \mathsf{true}$)

are concepts that do not exist in the Rainbow framework and the Znn.com exemplar. However, the authors of the framework quantify the fulfilment of the goals in their own way, using utility theory, which could be one approach to construct the quality function—as long as it adheres to the design rules that we presented in Appendix A.2. In Znn.com, the evaluation based on utility theory searches for some trade-off between the fulfilment of the quality objectives (the performance and the cost). If this evaluation determines that the fulfilment of the quality objectives is not within an accepted value, then this presents a trigger for self-adaptation. In response, a concrete strategy (i. e., action towards adaptation) is chosen.

Although parts of our theory are only implicitly implemented as part of Znn.com, the intuition that the authors have of self-adaptive systems aligns with the theoretical framework proposed as part of this dissertation. The utility theory-based solution from Znn.com can be mapped to our quality function with small modifications. Also, all the evaluation metrics from the existing exemplars that we summarized in Appendix A.1.2 can be modified and adjusted in a way so they adhere to the concrete rules for the construction of the Quality Function. This is another demonstration of the power and flexibility of our proposed theoretical framework. Namely, once the quality function can be measured and

constructed, it is the backbone for all the contributions that we make in this thesis. This helps to clearly pinpoint and differentiate between system functioning and system adapting (and self-adapting) and supports the specifications for engineering self-adaptive systems to differentiate these systems from those that are considered non-adaptive.

And lastly, it is important to point out that although the Rainbow framework is not built according to the MAPE-K conceptual model, the authors do differentiate between two layers in a self-adaptive system: the system layer (corresponding to the managed element) and the architecture layer (corresponding to the adaptation logic). In their framework, the adaptation layer is tailored using the system-specific adaptation knowledge, including the types and properties of components, behavioural constraints, and adaptation strategies (i. e., our introduced actions towards adaptation). This additionally validates our specification that the adaptation logic does not necessarily need to follow the MAPE-K conceptual model as long as it consists of a Knowledge component that is built according to the relevant aspects for the concrete self-adaptation (the system function, the adaptation goals, uncertain conditions and the actions towards adaptation).

A.6 Concluding remarks

Defining self-adaptive systems is challenging because this problem has a many-dimensional nature, and none of the existing literature has approached understanding and defining self-adaptive systems as exhaustively and comprehensively as we do in this paper. Towards this aim, in this appendix we proposed a theoretical framework for self-adaptive systems, which provides a holistic overview for understanding and defining self-adaptive systems. Throughout the whole thesis, we were motivated by the question of how self-adaptive systems differ from the "ordinary," non-adaptive systems. As a basis for the theoretical contributions in this appendix, we used:

- the informal definition of self-adaptive software by Laddaga [77]—concretely the specific part of his definition that was overlooked by the existing body of literature over the years, and
- our previous framing and formalization of system adaptation from Chapter 5.

The contributions of this appendix, and the overall dissertation, should ideally set the foundation for more constructive discussions on the existing and current works, hopefully supporting this research domain on a larger scale to complement the existing works and identify and focus on new future challenges and directions. To summarize, as part of this appendix:

1) we formally defined passive and active self-adaptive systems using functions that capture the behavioural aspect of these systems, besides also capturing the structural aspect of self-adaptive systems (through the proposed terminology and specifications),
2) elicited the minimum requirements for the adaptation logic in passive and active self-adaptive systems, as well as the overall requirements for self-adaptive systems as such and lastly,
3) we clearly specified the process of self-adaptation.

Alongside our theoretical contributions, we have discussed various emerging architectural implications based on our formalism, and we proposed a re-factored versions of the MAPE-K conceptual model for engineering passive and active self-adaptive systems. All our contributions are domain- and application-independent, meaning they are applicable for engineering any self-adaptive system, as we depicted throughout this appendix with examples from the robotics domain and software systems (e. g., Znn.com client-server system).

List of Figures

List of Tables

Bibliography

[1] Norm Abrahamson. *Aleatory variability and epistemic uncertainty*. 2007.

[2] Mai Abusair, Antinisca Di Marco, and Paola Inverardi. "Context-Aware Adaptation of Mobile Applications Driven By Software Quality and User Satisfaction". In: *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE. 2017, pp. 31–38.

[3] Frank José Affonso and Elisa Yumi Nakagawa. "A reference architecture based on reflection for self-adaptive software". In: *2013 VII Brazilian Symposium on Software Components, Architectures and Reuse*. Ieee. 2013, pp. 129–138.

[4] Mary B Alatise and Gerhard P Hancke. "A review on challenges of autonomous mobile robot and sensor fusion methods". In: *IEEE Access* 8 (2020), pp. 39830–39846.

[5] Robert Allen, Remi Douence, and David Garlan. "Specifying and analyzing dynamic software architectures". In: *International Conference on Fundamental Approaches to Software Engineering*. Springer. 1998, pp. 21–37.

[6] Konstantinos Angelopoulos et al. "Model predictive control for software systems with CobRA". In: *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE. 2016, pp. 35–46.

[7] P. Arcaini, E. Riccobene, and P. Scandurra. "Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation". In: *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, May 2015, pp. 13–23. DOI: 10.1109/SEAMS.2015.10.

[8] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. "Modeling and analyzing MAPE-K feedback loops for self-adaptation". In: *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE. 2015, pp. 13–23.

[9] Reza Asadollahi, Mazeiar Salehie, and Ladan Tahvildari. "StarMX: A framework for developing self-managing Java-based systems". In: *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE. 2009, pp. 58–67.

[10] K Suzanne Barber and Cheryl E Martin. "Agent autonomy: Specification, measurement, and dynamic adjustment". In: *Proceedings of the autonomy control software workshop at autonomous agents*. Vol. 1999. Citeseer. 1999, pp. 8–15.

[11] Nelly Bencomo, Amel Belaggoun, and Valerie Issarny. "Dynamic decision networks for decision-making in self-adaptive systems: a case study". In: *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE. 2013, pp. 113–122.

[12] Amel Bennaceur et al. “Mechanisms for leveraging models at runtime in self-adaptive software”. In: *Models@ run. time*. Springer, 2014, pp. 19–46.

[13] Gordon Blair, Nelly Bencomo, and Robert B France. “Models@ run. time”. In: *Computer* 42.10 (2009), pp. 22–27.

[14] Victor Braberman et al. “Morph: A reference architecture for configuration and behaviour self-adaptation”. In: *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*. 2015, pp. 9–16.

[15] Jeremy S Bradbury et al. “A survey of self-management in dynamic software architecture specifications”. In: *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*. 2004, pp. 28–33.

[16] James Patrick Brock. *The evolution of adaptive systems: The general theory of evolution*. Elsevier, 2000.

[17] Manfred Broy. “Formalizing Adaptivity, Dynamics, Context-Awareness, Autonomy”. unpublished. 2017.

[18] Manfred Broy and Ketil Stølen. *Specification and development of interactive systems: Focus on streams, interfaces, and refinement*. New York: Springer, 2001.

[19] Manfred Broy and Ketil Stølen. *Specification and development of interactive systems: Focus on streams, interfaces, and refinement*. New York: Springer, 2001.

[20] Manfred Broy et al. “Formalizing the notion of adaptive system behavior”. In: *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC)*. ACM, 2009, pp. 1029–1033. DOI: 10.1145/1529282.1529508.

[21] Yuriy Brun et al. “Engineering self-adaptive systems through feedback loops”. In: *Software engineering for self-adaptive systems*. Springer, 2009, pp. 48–70.

[22] Roberto Bruni et al. “A conceptual framework for adaptation”. In: *International Conference on Fundamental Approaches to Software Engineering*. Springer. 2012, pp. 240–254.

[23] Antonio Bucchiarone and Marina Mongiello. “Ten Years of Self-adaptive Systems: From Dynamic Ensembles to Collective Adaptive Systems”. In: *From Software Engineering to Formal Methods and Tools, and Back*. Springer, 2019, pp. 19–39.

[24] Javier Cámara, Bradley Schmerl, and David Garlan. “Software architecture and task plan co-adaptation for mobile service robots”. In: *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2020, pp. 125–136.

[25] Matteo Camilli, Raffaela Mirandola, and Patrizia Scandurra. “Runtime Equilibrium Verification for Resilient Cyber-Physical Systems”. In: *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE. 2021.

[26] Betty HC Cheng et al. “A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2009, pp. 468–483.

[27] Shang-Wen Cheng. “Rainbow: cost-effective software architecture-based self-adaptation”. PhD thesis. Carnegie Mellon University, 2008.

[28] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. “Architecture-based self-adaptation in the presence of multiple objectives”. In: *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*. 2006, pp. 2–8.

[29] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. “Evaluating the effectiveness of the rainbow self-adaptive system”. In: *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE. 2009, pp. 132–141.

[30] Mirko D’Angelo, Annalisa Napolitano, and Mauro Caporuscio. “CyPhEF: a model-driven engineering framework for self-adaptive cyber-physical systems”. In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 2018, pp. 101–104.

[31] Alberto Rodrigues Da Silva. “Model-driven engineering: A survey supported by the unified conceptual model”. In: *Computer Languages, Systems & Structures* 43 (2015), pp. 139–155.

[32] Rogério De Lemos et al. “Software engineering for self-adaptive systems: Research challenges in the provision of assurances”. In: *Software Engineering for Self-Adaptive Systems III. Assurances*. Springer, 2017, pp. 3–30.

[33] Frank Dellaert et al. “Monte carlo localization for mobile robots”. In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*. Vol. 2. IEEE. 1999, pp. 1322–1328.

[34] Arthur P Dempster. “A generalization of Bayesian inference”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 30.2 (1968), pp. 205–232.

[35] Armen Der Kiureghian and Ove Ditlevsen. “Aleatory or epistemic? Does it matter?” In: *Structural safety* 31.2 (2009), pp. 105–112.

[36] Jean Dezert and Albena Tchamova. “On the Validity of Dempster’s Fusion Rule and its Interpretation as a Generalization of Bayesian Fusion Rule”. In: *International Journal of Intelligent Systems* 29.3 (2014), pp. 223–252.

[37] Abdessalam Elhabbash et al. “Self-awareness in software engineering: A systematic literature review”. In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 14.2 (2019), pp. 1–42.

[38] Ahmed Elkhodary, Naeem Esfahani, and Sam Malek. “FUSION: a framework for engineering self-tuning self-adaptive software systems”. In: *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM. 2010, pp. 7–16.

[39] Naeem Esfahani, Ehsan Kouroshfar, and Sam Malek. “Taming uncertainty in self-adaptive software”. In: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 2011, pp. 234–244.

[40] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. "A formal approach to adaptive software: continuous assurance of non-functional requirements". In: *Formal Aspects of Computing* 24.2 (2012), pp. 163–186.

[41] Antonio Filieri et al. "Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements". In: *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE. 2011, pp. 283–292.

[42] Miguel de França Doria et al. "Using expert elicitation to define successful adaptation to climate change". In: *Environmental Science & Policy* 12.7 (2009), pp. 810–819.

[43] David Garlan, Bradley Schmerl, and Shang-Wen Cheng. "Software architecture-based self-adaptation". In: *Autonomic computing and networking*. Springer, 2009, pp. 31–55.

[44] David Garlan et al. "Rainbow: Architecture-based self-adaptation with reusable infrastructure". In: *Computer* 10 (2004), pp. 46–54.

[45] Andrew Gelman et al. "Part I Fundamentals of Bayesian Inference". In: *Bayesian Data Analysis*. 3. ed. CRC Press, 2014. Chap. 1, pp. 4–29. ISBN: 9781439840955 - 9781439840962.

[46] Ilias Gerostathopoulos et al. "Architectural homeostasis in self-adaptive software-intensive cyber-physical systems". In: *European Conference on Software Architecture*. Springer. 2016, pp. 113–128.

[47] Ilias Gerostathopoulos et al. "How do we Evaluate Self-adaptive Software Systems?" In: *arXiv preprint arXiv:2103.11481* (2021).

[48] Omid Gheibi, Danny Weyns, and Federico Quin. "Applying machine learning in self-adaptive systems: A systematic literature review". In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 15.3 (2021), pp. 1–37.

[49] Debanjan Ghosh et al. "Self-healing systems—survey and synthesis". In: *Decision support systems* 42.4 (2007), pp. 2164–2185.

[50] Ian Gorton, Yan Liu, and Nihar Trivedi. "An extensible, lightweight architecture for adaptive J2EE applications". In: *Proceedings of the 6th international workshop on Software engineering and middleware*. 2006, pp. 47–54.

[51] James P Gunderson and Louise F Gunderson. "Intelligence= autonomy= capability". In: *Performance Metrics for Intelligent Systems, PERMIS* (2004).

[52] M. Hachicha, R. B. Halima, and A. H. Kacem. "Formalizing compound MAPE patterns for decentralized control in self-adaptive systems". In: *2018 12th International Conference on Research Challenges in Information Science (RCIS)*. May 2018, pp. 1–10. DOI: 10.1109/RCIS.2018.8406680.

[53] M. Hachicha et al. "A correct by construction approach for modeling and formalizing self-adaptive systems". In: *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. May 2016, pp. 379–384. DOI: 10.1109/SNPD.2016.7515928.

[54] R. Haesevoets et al. “A formal model for self-adaptive and self-healing organizations”. In: *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, May 2009, pp. 116–125. DOI: 10.1109/SEAMS.2009.5069080.

[55] Svein Hallsteinsen et al. “A development framework and methodology for self-adapting applications in ubiquitous computing environments”. In: *Journal of Systems and Software* 85.12 (2012), pp. 2840–2859.

[56] Linda Hutcheon. *A theory of adaptation*. Routledge, 2012.

[57] M Usman Iftikhar et al. “Deltaiot: a real world exemplar for self-adaptive internet of things (artifact)”. In: *DARTS-Dagstuhl Artifacts Series*. Vol. 3. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017.

[58] Didac Gil De La Iglesia and Danny Weyns. “MAPE-K Formal Templates to Rigorously Design Behaviors for Self-Adaptive Systems”. In: *ACM Trans. Auton. Adapt. Syst.* 10.3 (Sept. 2015). ISSN: 1556-4665. DOI: 10.1145/2724719. URL: https://doi-org.eaccess.ub.tum.de/10.1145/2724719.

[59] Pooyan Jamshidi et al. “Machine learning meets quantitative planning: enabling self-adaptation in autonomous robots”. In: *SEAMS@ICSE*. ACM, 2019, pp. 39–50.

[60] Audun Jøsang. “A logic for uncertain probabilities”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 9.03 (2001), pp. 279–311.

[61] Audun Jøsang. *Subjective logic*. Springer, 2016.

[62] Audun Jøsang and Simon Pope. “Dempster’s rule as seen by little colored balls”. In: *Computational Intelligence* 28.4 (2012), pp. 453–474. DOI: 10.1111/j.1467-8640.2012.00421.x.

[63] Audun Jøsang and Francesco Sambo. “Inverting conditional opinions in subjective logic”. In: *20th International Conference on Soft Computing (Mendel 2014)* June (2014).

[64] Audun Jøsang, Dongxia Wang, and Jie Zhang. “Multi-source fusion in subjective logic”. In: *20th International Conference on Information Fusion, Fusion 2017 - Proceedings*. 2017. ISBN: 9780996452700. DOI: 10.23919/ICIF.2017.8009820.

[65] Jeffery Kephart. *Viewing Autonomic Computing through the Lens of Embodied Artificial Intelligence*. Keynote. 2021 IEEE/ACM 16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). 2021.

[66] Jeffrey O Kephart and David M Chess. “The vision of autonomic computing”. In: *Computer* 1 (2003), pp. 41–50.

[67] Barbara Kitchenham and Stuart Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Tech. rep. EBSE 2007-001. Keele University and Durham University Joint Report, 2007. URL: http://www.dur.ac.uk/ebse/resources/Systematic-reviews-5-8.pdf.

[68] Barbara Kitchenham et al. “Systematic literature reviews in software engineering–a tertiary study”. In: *Information and software technology* 52.8 (2010), pp. 792–805.

[69] A. Klarl. "Engineering Self-Adaptive Systems with the Role-Based Architecture of Helena". In: *2015 IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*. June 2015, pp. 3–8. DOI: 10.1109/WETICE.2015.32.

[70] Alessia Knauss et al. "ACon: A learning-based approach to deal with uncertainty in contextual requirements at runtime". In: *Information and software technology* 70 (2016), pp. 85–99.

[71] Samuel Kounev, Fabian Brosig, and Nikolaus Huber. *The Descartes Modeling Language*. Tech. rep. Institut für Informatik, University of Wuerzburg, Germany, 2014, p. 91.

[72] Samuel Kounev et al. "The notion of self-aware computing". In: *Self-Aware Computing Systems*. Springer, 2017, pp. 3–16.

[73] Christian Krupitzer et al. "A survey on engineering approaches for self-adaptive systems". In: *Pervasive and Mobile Computing* 17 (2015), pp. 184–206.

[74] Christian Krupitzer et al. "Comparison of Approaches for developing Self-adaptive Systems". In: (2018).

[75] Stefan Kugele, Ana Petrovska, and Ilias Gerostathopoulos. "Towards a Taxonomy of Autonomous Systems". In: *European Conference on Software Architecture*. to appear. Springer. 2021.

[76] Zadeh La. "Fuzzy sets". In: *Information and control* 8.3 (1965), pp. 338–353.

[77] R Laddaga. *Self-adaptive software DARPA BAA 98-12*. 1997.

[78] Philippe Lalanda, Julie A McCann, and Ada Diaconescu. "Autonomic computing". In: *Principles, Design and Implementation*. Springer, 2013.

[79] Thomas Leitch. "Adaptation, the genre". In: *Adaptation* 1.2 (2008), pp. 106–120.

[80] Rogério de Lemos, Holger Giese, Hausi A Müller, et al. "Software engineering for self-adaptive systems: A second research roadmap". In: *Software engineering for self-adaptive systems II*. Springer, 2013, pp. 1–32.

[81] Peter R Lewis et al. "A survey of self-awareness and its application in computing systems". In: *2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops*. IEEE. 2011, pp. 102–107.

[82] Taivo Lints. "The essentials of defining adaptation". In: *IEEE Aerospace and Electronic Systems Magazine* 27.1 (2012), pp. 37–41.

[83] Michael Luck and Mark d'Inverno. "A Formal Framework for Agency and Autonomy". In: *First International Conference on Multiagent Systems*. The MIT Press, 1995, pp. 254–260.

[84] Frank D Macías-Escrivá et al. "Self-adaptive systems: A survey of current approaches, research challenges and applications". In: *Expert Systems with Applications* 40.18 (2013), pp. 7267–7279.

[85] Martina Maggio. “Is this all about about handling unanticipated changes or about foreseeing what needs handling?” In: *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE. 2021, pp. 258–259.

[86] Sara Mahdavi-Hezavehi, Paris Avgeriou, and Danny Weyns. “A Classification Framework of Uncertainty in Architecture-Based Self-Adaptive Systems with Multiple Quality Requirements”. In: *Managing Trade-Offs in Adaptable Software Architectures* (2017), pp. 45–77.

[87] Doble J Meszaros and Jim Doble. “G. A pattern language for pattern writing”. In: *Proceedings of International Conference on Pattern languages of program design (1997)*. Vol. 131. 1997, p. 164.

[88] Gabriel Moreno et al. “DARTSim: an exemplar for evaluation and comparison of self-adaptation approaches for smart cyber-physical systems”. In: *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE. 2019, pp. 181–187.

[89] Susanne C Moser and Julia A Ekstrom. “A framework to diagnose barriers to climate change adaptation”. In: *Proceedings of the national academy of sciences* 107.51 (2010), pp. 22026–22031.

[90] Henry Muccini, Mohammad Sharaf, and Danny Weyns. “Self-adaptation for cyber-physical systems: a systematic literature review”. In: *Proceedings of the 11th international symposium on software engineering for adaptive and self-managing systems*. ACM, 2016, pp. 75–81.

[91] Angelika Musil et al. “Patterns for self-adaptation in cyber-physical systems”. In: *Multi-disciplinary engineering for cyber-physical production systems*. Springer, 2017, pp. 331–368.

[92] Peyman Oreizy et al. “An architecture-based approach to self-adaptive software”. In: *IEEE Intelligent Systems and Their Applications* 14.3 (1999), pp. 54–62.

[93] John Allen Paulos. “The mathematics of changing your mind”. In: *New York Times (US)* (2011).

[94] Ana Petrovska. “Self-Awareness as a Prerequisite for Self-Adaptivity in Computing Systems”. In: *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE. 2021, pp. 146–149.

[95] Ana Petrovska and Alexander Pretschner. “Learning Approach for Smart Self-Adaptive Cyber-Physical Systems”. In: *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE. 2019, pp. 234–236.

[96] Ana Petrovska et al. “Defining adaptivity and logical architecture for engineering (smart) self-adaptive cyber–physical systems”. In: *Information and Software Technology* 147 (2022), p. 106866.

[97] Ana Petrovska et al. “Knowledge aggregation with subjective logic in multi-agent self-adaptive cyber-physical systems”. In: *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2020, pp. 149–155.

[98] Ana Petrovska et al. “Run-time reasoning from uncertain observations with subjective logic in multi-agent self-adaptive cyber-physical systems”. In: *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE. 2021, pp. 130–141.

[99] Michiel Provoost and Danny Weyns. “DingNet: a self-adaptive internet-of-things exemplar”. In: *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE. 2019, pp. 195–201. URL: https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/hogna/.

[100] Harald Psaier and Schahram Dustdar. “A survey on self-healing systems: approaches and systems”. In: *Computing* 91.1 (2011), pp. 43–73.

[101] Federico Quin, Danny Weyns, and Omid Gheibi. “Decentralized Self-Adaptive Systems: A Mapping Study”. In: *arXiv preprint arXiv:2103.09074* (2021).

[102] Federico Quin et al. “Efficient Analysis of Large Adaptation Spaces in Self-Adaptive Systems using Machine Learning.” In: *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)* (2019), pp. 1–12.

[103] Nauman A Qureshi, Ivan J Jureta, and Anna Perini. “Requirements engineering for self-adaptive systems: Core ontology and problem statement”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2011, pp. 33–47.

[104] Andres J Ramirez, Adam C Jensen, and Betty HC Cheng. “A taxonomy of uncertainty for dynamically adaptive systems”. In: *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press. 2012, pp. 99–108.

[105] Romain Rouvoy et al. “Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments”. In: *Software engineering for self-adaptive systems*. Springer, 2009, pp. 164–182.

[106] Vineet Saini, Qiang Duan, and Vamsi Paruchuri. “Threat modeling using attack trees”. In: *Journal of Computing Sciences in Colleges* 23.4 (2008), pp. 124–131.

[107] Mazeiar Salehie and Ladan Tahvildari. “Self-adaptive software: Landscape and research challenges”. In: *ACM transactions on autonomous and adaptive systems (TAAS)* 4.2 (2009), p. 14.

[108] Ronny Seiger et al. “Toward a framework for self-adaptive workflows in cyber-physical systems”. In: *Software & Systems Modeling* 18.2 (2019), pp. 1117–1134.

[109] Glenn Shafer. *A mathematical theory of evidence*. Vol. 42. Princeton university press, 1976.

[110] Seung Yeob Shin et al. “Dynamic adaptation of software-defined networks for IoT systems: A search-based approach”. In: *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2020, pp. 137–148.

[111] Howard Shrobe and Robert Laddaga. *Self Adaptive Software*.

[112] Ph. Smets. “Practical Uses of Belief Functions”. In: *Uncertainty in Artificial Intelligence 15. UAI99* (1999), pp. 612–621.

[113] Rens W. Van Der Heijden, Henning Kopp, and Frank Kargl. “Multi-Source Fusion Operations in Subjective Logic”. In: *2018 21st International Conference on Information Fusion, FUSION 2018* (2018), pp. 1990–1997. DOI: 10.23919/ICIF.2018.8455615. arXiv: arXiv:1805.01388v1.

[114] Norha M Villegas et al. “A framework for evaluating quality-driven self-adaptive software systems”. In: *Proceedings of the 6th international symposium on Software engineering for adaptive and self-managing systems*. 2011, pp. 80–89.

[115] Thomas Vogel, Andreas Seibel, and Holger Giese. “The role of models and megamodels at runtime”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2010, pp. 224–238.

[116] Danny Weyns. *An Introduction to Self-adaptive Systems: A Contemporary Software Engineering Perspective*. John Wiley & Sons, 2020.

[117] Danny Weyns. “Software engineering of self-adaptive systems”. In: *Handbook of Software Engineering*. Springer, 2019, pp. 399–443.

[118] Danny Weyns. “Software engineering of self-adaptive systems: an organised tour and future challenges”. In: *Chapter in Handbook of Software Engineering* (2017).

[119] Danny Weyns and Tanvir Ahmad. “Claims and evidence for architecture-based self-adaptation: a systematic literature review”. In: *European Conference on Software Architecture*. Springer. 2013, pp. 249–265.

[120] Danny Weyns, Sam Malek, and Jesper Andersson. “FORMS: Unifying reference model for formal specification of distributed self-adaptive systems”. In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 7.1 (2012), pp. 1–61.

[121] Danny Weyns et al. “A survey of formal methods in self-adaptive systems”. In: *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*. 2012, pp. 67–79.

[122] Danny Weyns et al. “Claims and supporting evidence for self-adaptive systems: A literature study”. In: *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE. 2012, pp. 89–98.

[123] Danny Weyns et al. “On patterns for decentralized control in self-adaptive systems”. In: *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 76–107.

[124] Jon Whittle et al. “Relax: Incorporating uncertainty into the specification of self-adaptive systems”. In: *2009 17th IEEE International Requirements Engineering Conference*. IEEE. 2009, pp. 79–88.

[125] Claes Wohlin. “Guidelines for snowballing in systematic literature studies and a replication in software engineering”. In: *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 2014, pp. 1–10.

[126] E. Woods. “Software Architecture in a Changing World”. In: *IEEE Software* 33.6 (Nov. 2016), pp. 94–97. ISSN: 0740-7459. DOI: 10.1109/MS.2016.149.

[127] Lotfi A Zadeh. “On the definition of adaptivity”. In: *Proceedings of the IEEE* 51.3 (1963), pp. 469–470.

[128] Lotfi A. Zadeh and Anca Ralescut. “On the Combinability of Evidence in the Dempster-Shafer Theory”. In: (2013). arXiv: 1304.3119v1.

[129] Ji Zhang and Betty H. C. Cheng. “Model-based development of dynamically adaptive software”. In: *Proceeding of the 28th international conference on Software engineering - ICSE ’06*. ACM Press, 2006. DOI: 10.1145/1134285.1134337.