THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Service Robotics Software Engineering

SERGIO GARCÍA

UNIVERSITY OF GOTHENBURG

Division of Interaction Design and Software Engineering
Department of Computer Science & Engineering
University of Gothenburg
Gothenburg, Sweden, 2021

Service Robotics Software Engineering

SERGIO GARCÍA

ISBN 978-91-8009-410-8
ISSN 0346-718X

Technical Report No 202D
Department of Computer Science & Engineering
Division of Interaction Design and Software Engineering
University of Gothenburg
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2021.

“The trouble with having an open mind, of course, is that people will insist on coming along and trying to put things in it.”

- Terry Pratchett

Abstract

Context. Robots are increasingly becoming involved in our lives and currently, teams of service robots cooperate to support humans by performing useful, repetitive, or dangerous tasks. However, engineering the robots' software to ensure their robustness and autonomy has become a bottleneck in their development, mainly due to the inherent complexity of the domain. Service robots typically operate in partially constrained environments—often populated by humans—and provide many services for which they require a blend of hardware and software components. This requires developers to possess a diversity of expertise stemming from different disciplines. Building software with the ability to be changed, customized, or configured to fit different contexts, environments, or purposes is one of the core means to address these challenges. Unfortunately, software engineering has been mostly considered an auxiliary concern in the robotics community in the past.
Objective. The objective of this thesis is to investigate practices and challenges in service robotics software engineering to provide guidance for practitioners and researchers as well as to engineer solutions for some of the challenges we identified.
Method. Our research methodology combines knowledge-seeking and solution-seeking research. The former's aim is to learn about a specific phenomenon; in this case, the state-of-the-art and -practice in service robotics software engineering. With this aim, we conducted interviews with practitioners, an online questionnaire, and a systematic literature review. The goal of solution-seeking research is to solve problems within a specific context, for which we designed solutions. The research for this thesis has been conducted in the context of a European project in collaboration with industrial partners. This allowed us to identify software engineering problems in the service robotics industry and validate our solutions in real-world scenarios and robots.
Results. We outline with our empirical results the current practices, characteristics, and challenges of service robotics software engineering. We found that the most pressing challenges are validating and achieving the robustness of autonomous systems, especially those whose behavior and control systems are intended to be changeable, configurable, or customizable. The solutions we engineered consist of a software architecture for multi-robot applications (SERA), a domain-specific language for robot mission specification (PROMISE), and a software component that is able to synthesize plans considering only partial knowledge of a robotic application (MAPmAKER). We integrated our engineered solutions into an industry-validated robotic framework that is structured by the premises established for SERA and whose composing modules were validated in terms of flexibility, robustness, and efficiency.
Conclusions. Our empirical contributions help to enhance the current empirical understanding in the domain of service robotics software engineering and its practices, characteristics, and challenges with the aim of providing guidance for practitioners and researchers. Our engineered solutions consist of methods and tools whose aim is to support practitioners and researchers in building well-engineered robotic applications and in operating them.

Keywords

Autonomous and (Self-)Adaptive Systems, Service Robots, Robotics Software Engineering, Software Architecture, Empirical Research, Model-Driven Engineering, Domain-Specific Languages, Variability.

Acknowledgment

I am deeply grateful to many people for their support during the last years. First and foremost, I would like to express my gratitude to my supervisor Patrizio Pelliccione for his priceless mentorship, guidance, and encouragement. Your continuous feedback and inspiration have made this thesis possible, for this and your patience I remain in debt. I also want to express my gratitude to my co-supervisor Thorsten Berger for his great insight, knowledge, and assistance as well as to my examiner Gerardo Schneider for his advice while following my research.

Working in such a great work environment has made conducting my research through these years much more pleasant. For this, I want to thank all my colleagues at the Interaction Design and Software Engineering division, including all the administrative staff. I am especially grateful to my fellow Ph.D. students for their friendship and experiences, and more concretely to my former office mates Pier, Katja, and Rebekka. I hope all the best for you in your bright future.

I want to express my gratitude to all the participants in my studies who decided to lend me a hand unselfishly, especially those from PAL Robotics, the Bosch Center of Artificial Intelligence, and Blue Ocean Robotics.

I am grateful to all the partners and friends I made in the context of Co4Robots for all those Integration Weeks of hard work and late dinners.

A big *thank you* goes to all my co-authors, my research would have not been possible without you. I would like to especially thank Claudio Menghi, Daniel Strüber, and Davide Brugali for all the hard work, support, and fruitful discussions.

I would further like to thank my friends in Spain, Sweden, and the rest of the world. Thank you for everything we have been through together all these years.

To my family; father, mother, and brother, thank you from the bottom of my heart. Thank you for helping me shape my personality and for unconditionally believing in me. You made me who I am and are always there for me.

Finally, I would like to express my utmost gratitude to my life partner, Arantxa, for being a constant source of motivation through all these years. I am here because of you and your love, this thesis is yours as much as it is mine.

This work was partly supported by the EU H2020 Research and Innovation Program under GA No. 731869 (Co4Robots).

List of Publications

Appended publications

This thesis is based on the following publications:

[A] S. García, D. Strüber, D. Brugali, T. Berger, and P. Pelliccione.
"Robotics Software Engineering: A Perspective from the Service Robotics Domain."
Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'20), 2020.

[B] S. García, D. Strüber, D. Brugali, A. Di Fava, P. Pelliccione, and T. Berger.
"Software Variability in Service Robotics."
Submitted and under revision *to the special issue "Software Product Lines and Variability-rich Systems" in the journal Empirical Software Engineering (EMSE), 2021.*

[C] S. Dragule, S. García, T. Berger, and P. Pelliccione.
"Languages for Specifying Missions of Robotic Applications."
Book chapter in "Software Engineering for Robotics," Springer, 2021.

[D] S. García, C. Menghi, P. Pelliccione, T. Berger, and R. Wohlrab.
"An Architecture for Decentralized, Collaborative, and Autonomous Robots."
Proceedings of the IEEE International Conference on Software Architecture (ICSA'18), 2018.

[E] S. García, P. Pelliccione, C. Menghi, T. Berger, and T. Bures.
"High-Level Mission Specification for Multiple Robots."
Proceedings of the ACM SIGPLAN International Conference on Software Language Engineering (SLE'19), 2019.

[F] C. Menghi, S. García, P. Pelliccione, and J. Tumova.
"Multi-robot LTL planning under uncertainty."
Proceedings of the International Symposium on Formal Methods, 2018.

[G] M. Logothetis, G. Karras, K. Alevizos, C. Verginis, P. Roque, K. Roditakis, A. Makris, S. García, A. Di Fava, P. Pelliccione, P. Schillinger, A. Argyros, K. Kyriakopoulos, and D. Dimarogonas.
"A Decentralized Framework for Efficient Cooperation of Heterogeneous Robotic Agents."

Special issue "Emerging Paradigms for Robotic Manipulation: from the Lab to the Productive World" in the journal IEEE Robotics & Automation Magazine, 2021.

[H] P. Schillinger, S. García, A. Makris, K. Roditakis, M. Logothetis, K. Alevizos, W. Ren, P. Tajvar, P. Pelliccione, A. Argyros, K. Kyriakopoulos, and D. Dimarogonas.
"Adaptive Heterogeneous Multi-Robot Collaboration from Formal Task Specifications."
Submitted and under revision *to the special issue "Intelligent Autonomous Systems" in the Robotics and Autonomous Systems Journal, 2021.*

Other publications

The following publications were published during my PhD studies but are not appended due to contents overlapping that of appended publications or contents not related to the thesis.

[a] S. García.
"Effective Engineering of Multi-Robot Software Applications."
Proceedings of the IEEE/ACM 40th International Conference on Software Engineering: Companion (Doctoral Symposium, ICSE'18), 2018.

[b] S. García, C. Menghi, and P. Pelliccione.
"MAPmAKER: Performing Multi-Robot LTL Planning Under Uncertainty."
Proceedings of the 2nd International Workshop on Robotics Software Engineering (RoSE'19), 2019.

[c] C. Menghi, S. García, P. Pelliccione, and J. Tumova.
"Towards Multi-Robot Applications Planning Under Uncertainty."
Proceedings of the IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), 2018.

[d] S. García, P. Pelliccione, C. Menghi, T. Berger, and T. Bures.
"PROMISE: High-Level Mission Specification for Multiple Robots."
Tool demo paper at ICSE 2020. Proceedings of the IEEE/ACM 42th International Conference on Software Engineering: Companion (ICSE-Companion), 2020.

[e] S. García, D. Strüber, D. Brugali, A. Di Fava, P. Schillinger, P. Pelliccione, and T. Berger.
"Variability Modeling of Service Robots: Experiences and Challenges."
Proceedings of the 13th International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS'19), 2019.

Research Contribution

In all papers for which I was the first author, I assumed the responsibility of being the driving force and writing major parts of the publications.

For paper A, I was the main responsible for the data collection (conducting 18 interviews and designing an online survey questionnaire with 156 responses), data analysis and triangulation, discussion, as well as writing.

In Paper B, I conducted the 11 interviews with practitioners and contributed to the Systematic Literature Review (SLR) by analyzing 12 of the papers. In that regard, I also held weekly meetings with the author who mainly conducted the SLR to provide guidance in form of sanity checks. In this paper, I was the main responsible for data analysis and triangulation, discussion of findings, and writing.

In Paper C, my contribution is twofold. First, I contributed by counseling the first author while investigating robotics-tailored domain-specific languages and integrated development environments. Second, I wrote entire sections of the paper, including the introduction, the section for programming languages, the introductory part for mission specification, and everything related to PROMISE.

For paper D, I contributed by designing the software architecture and implementing the platform that realizes it. I collected and analyzed data from various focus groups I participated in and from two informal surveys I distributed among the partners of a European project to validate and iteratively update the architecture's design. I also contributed to the design of the research method, conducting a literature review, and performing experiments in simulation and with real robots for the architecture's validation.

For paper E, I contributed by helping to conceive and develop the language of our domain-specific language PROMISE together with two of the paper authors. I later implemented PROMISE into a toolchain. I also performed the experimentation part of the research, testing PROMISE in simulation and with five different robotic platforms in real-world scenarios. For the validation of the research, I also conducted the latest of our user studies with six participants. I was also the main responsible for the discussion and writing of the paper.

For paper F, I mainly contributed to the implementation of MAPmAKER in MATLAB, the experimentation, validation, discussion, and writing of results.

My contribution to paper G was helping with the design and conduction of experiments, focusing on providing methods and tools to ease the integration of software components. I also contributed to the writing of the paper.

In paper H I contributed by helping with the design of the decentralized framework as well as conceiving and implementing the used domain-specific mission specification patterns based on our findings from papers D and E, respectively. I also helped by writing parts of the paper.

Contents

Chapter 1

Introduction

Robots are becoming increasingly involved in our everyday life to support us in monotonous, menial, or dangerous tasks. Contrary to automatized, reprogrammable manipulators that are typically applied to industrial environments like assembly lines (i.e., *industrial robots*), *service robots* are a rising robotics domain with broad applications in many fields—e.g., logistics, healthcare, telepresence, maintenance, domestic tasks, education, and entertainment [121]. A service robot is "*a type of robot that performs useful tasks for humans or equipment excluding industrial automation applications*" [71]. Service robots range from robots that automate housework like vacuum cleaning or lawn mowing to professional service robots used by companies to perform specialized tasks (e.g., supplies transportation in logistics contexts).

Moreover, service robots have demonstrated being of tremendous utility in situations when human lives are at risk. They are currently being used in the fight against COVID-19. Relevant examples for the latter issue are disinfecting robots able to kill bacteria and viruses in human-populated areas such as hospitals, government buildings, offices, hotels, airports, and universities,[1] as well as delivery robots that transport items around hospitals, reducing the time pressure on hospital staff and helping with safety distancing.[2] In this context, several market domains have noticed the convenience of implementing robotic applications to their environments since they are not constrained by travel bans or health issues and help to keep social distancing.[3] COVID-19 has indeed boosted the robotics market, and more especially for disinfection and logistics robots. This growing tendency sums up to a previously reported growth: the sales value of professional service robots was estimated to be increased by 32% to USD11.2 billion worldwide between 2018 and 2019.[4]

Robots are cyber-physical systems that blend hardware and software components to interact with their environment [22, 25]. As autonomous robots have started to become commodities, the service robotics industry community has realized that the cost of developing new robotic products is significantly related to the complexity of de-

[1] https://www.forbes.com/sites/richblake1/2020/04/17/in-covid-19-fight-robots-report-for-disinfection-duty

[2] https://www.mvpromedia.com/article/robots-to-assist-fight-against-covid-19-in-hospitals/

[3] https://ifr.org/img/worldrobotics/Executive_Summary_WR_2020_Service_Robots.pdf

[4] https://ifr.org/ifr-press-releases/news/service-robots-record-sales-worldwide-up-32

veloping software control systems that are flexible enough to be changed in order to fit different contexts, environments, or purposes [25]. The development and integration of hardware, software, and environmental components that fit these requirements give rise to the significant complexity of service robotic systems. The management of such complexity calls for systematic software engineering practices—as they have been successfully applied to other cyber-physical domains, e.g., automotive or aeronautics—that could impact the robotics marketplace by stimulating component supply-chains [17]. The rationale is that software engineering research and practices strive to develop and validate methods, technologies, and tools that can practically be useful in supporting industry [112]. We define robotics software engineering as the systematic application of engineering practices, methods, and tools to the development of software for robotics.

The development of a robot requires expertise stemming from diverse areas that need to be integrated into a common platform, including mechatronics, computer vision, and control theory. This also implies that many developers involved in the development of a robot are not necessarily software engineers; however, developing software applications without applying software engineering practices, methods, and tools gives rise to challenges such as quality assurance, validation, and integration. Furthermore, service robotics poses domain-specific challenges, as the ones implied by the complexity caused by multi-robot applications [76] and human-robot interaction [60, 54]—besides the challenges imposed by operating cyber-physical [100] real-time [88] systems. In this thesis, we identify challenges for practitioners in service robotics software engineering, e.g., integrating heterogeneous software components, validating and instilling robustness in robotic applications.

Concretely, we identified that managing *variability*—"*the ability of a software artifact to be changed in order to fit different contexts, environments, or purposes*" [47]—is a core challenge that affects service robotics companies. Robotics software needs to account for a diversity of hardware, operating environments, and customer demands, without being redeveloped from scratch. As such, robot developers need to manage variability in their software using so-called variability mechanisms, which are implementation techniques to realize variation points. However, while drivers of variability—i.e., hardware diversity, environment uncertainty, or the different purposes and functions of robots—and variability mechanisms are reasonably well understood in other domains, that is not the case for the vibrant domain of (autonomous) service robots. If not managed properly, variability in complex systems may lead to failures, including drops of performance or crashing of the system.

The overall objective of this thesis is to investigate practices and challenges in service robotics software engineering to provide guidance for practitioners and researchers as well as to engineer solutions for some of the challenges we identified. This objective is divided into two subgoals:

G1: To provide empirical insights of the current state-of-the-art and -practice of service robotics software engineering that lead to establishing effective practices, methods, and tools for engineering service robotics applications.

G2: To identify, explore, and engineer solutions for some of the practical problems in the service robotics domain from our findings in **G1**.

To fulfill our objectives we conducted a combination of knowledge-seeking research—aiming to learn about robotics software engineering—and solution-seeking research—aiming to solve identified practical problems for which we engineered solutions [124, 125].

Outline. The remainder of this chapter continues as follows. In Section 1.1 we introduce the required background for the thesis. We then describe in further detail the goals of this thesis in Section 1.2 and the methodology we followed to accomplish them in Section 1.3. Section 1.4 provides a summary of the contributions in the form of appended papers to the thesis. In Section 1.5 we discuss our findings. We conclude and provide our next steps as future work in Section 1.6.

1.1 Background and Related Work

In this section, we introduce important concepts that will be mentioned and used throughout the thesis. Concretely, we present concepts within the topics of *service robotics*, *mission specification*, *the Robot Operating System*, *software architectures*, and *variability and its management*.

1.1.1 Robots

The first part of this section gives a short notion of the history of robotics and the second introduces technical terms required for this thesis.

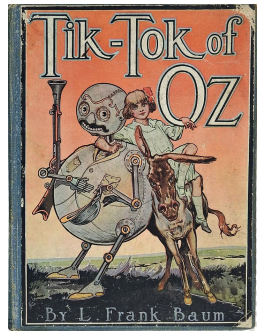
1.1.1.1 A Bit of History

The idea of human-looking animated entities that serve, protect, or interact with human beings goes back more than two millennia. The first possible example of such an idea appears in Greek mythology in the form of Talos, a sort of giant automaton made of metal (see Figure 1.1c) built to protect Crete from pirates and invaders. Many centuries later, the first artificial human makes an appearance in the 19th century by the hands of Edward S. Ellis in his book "The Steam Man of the Prairies" (Figure 1.1a). In this book, a "*colossal, steam-powered ma*n" is manufactured by a teenager and is used to carry a carriage in various adventures. Later in the early 20th century, L. Frank Baum introduces Tik-Tok (see Figure 1.1b), a character that has been termed "the prototype robot" [38] 10 years before this term was coined. As opposed to previous animated humanoids from literature, Tik-Tok is described as a mechanical man with intelligence and the power of reasoning.

The term *robot* was coined in 1921 by the Czech writer Karel Čapek in his play Rossum's Universal Robot (R.U.R.). In this satire, robots are artificial people who perform all unpleasant manual labor. Indeed, robot is a variation of *robota*, which in Czech means "servitude" or "forced labor."

After some attempts of building robots as machines able to learn and reproduce some basic tasks such as giving a speech or writing [62], the first *industrial robot* is developed in 1959 by George Devol and Joseph Engelberger [84, 65]. Industrial robots have been since the second half of the 20th century the most widespread type of robot. The International Organization for Standardization (ISO) 8373:2012 [73] defines an industrial robot as: "*an automatically controlled, reprogrammable multipurpose manipulator programmable in three or more axes*." This type of robot has been successfully applied in industrial automation [84], where they perform automated tasks in highly constrained environments. The domain of industrial robots has reached a high state of maturity, with a special focus on the automotive industry, being a driver for superior productivity and earnings [66].

(a) Cover of Edward S. Elis' 1868 book *The Steam Man of the Prairies*.



(b) Cover of L. Frank Baum's 1907 book *Ozma of Oz*.

(c) Scene from Don Chaffey's 1963 film *Jason and the Argonauts*.

Figure 1.1: Examples of artificial humans from mythology and modern literature.[5]

In the last decades, other types of robots have appeared in the scene [65]. Nowadays robots are complex systems that embody a blend of hardware and software and that interact with the environment, a.k.a., cyber-physical systems. The ISO 8373:2012 standard describes a robotic control system as a "*set of logic control and power functions that allows monitoring and control of the mechanical structure of the robot and communication with the environment*" [73]. Nowadays robots are highly versatile machines that are able to achieve a surprising variety of tasks for which they need to operate in various environments. Examples of applications for contemporaneous robots help with, among others, enhancing transportation safety, reducing agricultural pesticide use, and improving public safety and crime-fighting efficacy [25, 67].

1.1.1.2 Service Robotics

In this thesis, we focus on service robots, which are defined by the ISO 8373:2012 as a type of robot that supports humans by performing useful tasks [73]. As opposed to industrial robots, service robots are called to operate in partially constrained environments due to their versatility [17, 121]. In such scenarios, service robots will need to cope with uncertainty and possibly with humans, with which robots could be required to cooperate [121]. In our studies, we further classify service robots into two subclasses:

- *Professional service robots*, according to the ISO 8373:2012 [73], usually re-

[5]The figures are in the public domain and attribution is not legally required.

quire a trained operator for their operation and are applied to specific commercial tasks. As they are mostly targeted to end-user domains, they typically do not require programming skills from the operator to operate the robots although they require an expert to program the robots' functionalities beforehand. Examples of professional service robots are the UVD[6] and Stockbot[7] robots.

- *Research platforms* are a type of robot whose special features make it adequate for research. They may be used to assess the efficiency of robotics software and newly developed applications as well as to create proofs-of-concept to validate scientific contributions. These robots are mainly targeted at researchers and developers and therefore their operation usually requires knowledge of programming languages and robotics. Examples of research platforms are the TurtleBot2[8] and the TIAGo[9] robots.

The ISO 8373:2012 [73] defines the *operator* as the person designated to start, monitor, and stop the intended operation of a robot. In other words, the operator is the person in charge of commanding tasks and missions (explained shortly) to the robots. For the remainder of the thesis, we differentiate between technical and non-technical operators. *Technical operators* are knowledgeable on programming languages or robotics and are able to use advanced mechanisms for mission specification—e.g., behavior trees, finite-state machines, general-purpose languages. On the other hand, *non-technical operators* do not necessarily possess knowledge of programming or robotics and therefore mechanisms such as user-friendly graphical interfaces may be more suitable for them.

The *environment* is defined as the space and situation where the robots operate. Examples of environments treated throughout the thesis are hospitals, warehouses, farms, universities, and offices, each of them posing different requirements and constraints. We refer to the ISO/PAS 21448:2019 [72] to define *events* as phenomena that may occur in the environment. Robotic applications need to react to those events based on predefined *triggers* and *adapt* their behavior accordingly [105].

The mechanical embodiment and the sensors and actuators a robot equips provide it with a set of *capabilities*. The locomotion system of a mobile robot allows it to navigate, a robotic manipulator can grasp objects thanks to its robotic arm, and a robot equipped with a camera can "see" the environment where it operates. In turn, *skills* are the programmed *actions* a robot can perform, based on its capabilities. The ISO/PAS 21448:2019 introduces the term *action* as "*an atomic behaviour that is executed by any actor in a scene*" [72]. Skills are often implemented as software modules or *components* by experts. Examples of robotic *skills* range from the software program used by a robot to grasp an object to controllers for sensors and actuators. Another important term used throughout the thesis is *mission*, described as a coordinated combination of skills and actions that express the desired behaviors of the robots. This term has been introduced and used in literature, by among others, García et al. [48], Ghzouli et al. [52], Menghi et al. [97], and Menghi et al. [95].

An example of a mission is: "*A robot r*1 *operating in a hospital consisting of a number of rooms and corridors must reach room*2 *and disinfect it*." The terms *mission* and *task* are often used synonymously in the literature, even in reference

[6] https://www.uvd-robots.com
[7] https://pal-robotics.com/robots/stockbot/
[8] https://www.turtlebot.com/turtlebot2/
[9] https://pal-robotics.com/robots/tiago/

documents such as the H2020 Robotics Multi-Annual Roadmap (MAR) [121]. The ISO 8373:2012 [73] defines a task program as the "*set of instructions for motion and auxiliary functions that define the specific intended task of the robot or robot system*." To make a clearer definition of these terms, we refer to tasks as repetitive and simpler processes than missions. Therefore, tasks are repetitive and simple coordinated robotic behaviors that are realized as a combination of skills. A mission can be constructed by composing several tasks; for instance, in the example above, navigating to target locations and disinfection routines are considered tasks in this thesis.

1.1.2 Mission Specification

Future robotic applications will include multi-purpose mobile robots that may be configured or specified by (possibly non-technical) operators to perform missions of everyday life, as interpreted by the MAR [121]. For example, the specification of the mission introduced above in which a robot disinfects rooms in a hospital can be performed either programmatically in a General-Purpose Language (GPL), via Domain-Specific Languages (DSLs), or by using logical languages [55, 120, 113, 90]. In the following, we describe technologies that have been used for mission specification by the robotics community.

1.1.2.1 Behavioral modeling

A wide range of languages and formalisms for modeling and reasoning about behaviors has been developed, many of these have also been used for describing the behavior of robots. Statecharts [59] have been successfully applied to describe robotic missions [15, 126, 78, 6]. They are an extension of state machines and state diagrams, with a graphical syntax and formal semantics. This type of diagram requires a detailed definition of the steps a robot (or a team of robots) must perform in order to achieve a mission, which may become complicated in practical cases.

Another formalism is Petri Nets [107], which in the past have been successfully applied to develop robotic applications [133, 140]. In particular, Petri Net Plans (PNPs) [140] were developed to allow developers to describe plans for teams of robots. Despite their expressive power, PNPs require a precise definition of every action a robot is requested to perform.

Behavior trees [29] is a formalism originally conceived to define the behavior of autonomous non-player characters (NPCs) in computer games [70]. The idea behind the formalism was to support NPCs to behave in a reactive manner and to make decisions in complex and unpredictable environments [61, 70]. This way of behaving is similar to that of robots, which makes behavior trees an attractive formalism to define robotic behavior (i.e., mission specification). Furthermore, behavior trees enable modularization and reusability in mission specification and provide a purportedly easy-to-understand hierarchical structure, able to represent layers of behavior [29]. Compared to hierarchical state machines [139], whose usage simplifies the modeling and specification of complex and dynamic scenarios compared to simple state machines [61], behavior trees are considered more human-readable and easy to understand [29].

1.1.2.2 Domain-Specific Languages

Programming robots and their behaviors is complex, challenging, and error-prone for reasons ranging from the variety of capabilities robots may provide, their interaction

with the real world, and the fact that they constitute safety-critical systems. One of the biggest factors that trigger this complexity lies in the fact that robot programming is a knowledge-intensive task. That is, programming a robot to execute a, for example, transportation mission[10] requires knowledge from several domains, including control theory, computer vision, mechatronics, and software engineering.

As explained by Schneider et al. [118], approaches based on Model-based Development (MBD) and the Model-Driven Engineering (MDE) [117] paradigm permit exploiting the knowledge for systematic software development. For instance, the study of Bhave et al. [14] applies MBD to heterogeneous models using architectural views of cyber-physical systems. Mappings among those views are used to ensure the structural consistency of the models. The study is extended by Rajhans et al. [110], where they propose a framework that employs behavioral semantics to support heterogeneous multi-model development of cyber-physical systems.

The MDE paradigm uses the concept of meta-models to design, develop, and investigate abstractions over a certain domain in a layered fashion. Looking at the 4-level metamodel hierarchy of the OMG[11] [19] (see Fig. 1.2), the level M2 represents the metamodel level. This level formalizes the domain knowledge, including domain concepts and their names, relations among them, and their properties. This level of the hierarchy can be implemented or represented by a Domain-Specific Language (DSL) [46], which according to Van Deursen et al. [131], can be defined as "*a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain*." These languages can map the domain knowledge with familiar terminology and syntactical constructs to the domain practitioners, which makes contemporary DSLs more user-friendly and tailored to the robotics domain than General-Purpose Languages. Thus, DSLs help improving productivity for developers while at the same time improving communication with domain experts [46]. Furthermore, metamodels for a domain are stable whereas domain models may differ, which enhances the design and development of software by and for reuse.

In fact, MDE has been identified by the H2020 Multi-Annual Robotics Roadmap (MAR) [121] as a core technology to support developers when designing robotic systems. The MAR makes special emphasis on DSLs, which are required to achieve a separation of roles in the robotics domain while also improving composability and system integration while addressing non-functional properties. The research community has already worked in the last years on the use of languages for the development of robotic software systems [26, 117, 115, 122]. They have also been used to reason on the robot's behaviors through automated reasoning techniques such as simulation, model checking, and theorem proving, as compiled in the surveys conducted by Nordmann et al. [103, 104].

1.1.2.3 Temporal Logic

Classical Boolean logic allows formulating specifications φ over a set of propositions Π. Each proposition $\pi \in \Pi$ can either be true $\top$ or false $\bot$. Temporal logic extends the Boolean logic by operators with temporal semantics. This additional expressiveness makes temporal logics useful for specifying the desired behavior of autonomous systems, effectively capturing the temporal evolution of these systems.

[10] https://www.youtube.com/watch?v=rfljWQomk-I
[11] https://www.omg.org

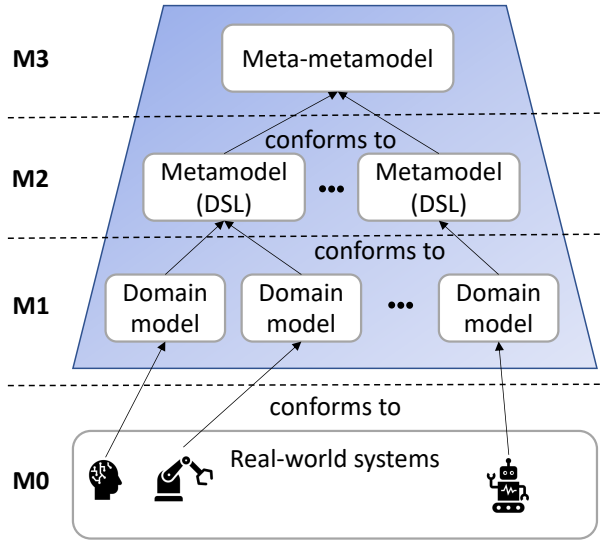


Figure 1.2: OMG's 4-level metamodel hierarchy, as adapted from Klotzbücher et al. [79].

The ability to prove formal properties on the behavior of a cognitive robotic system, especially related to safety, is pivotal to their introduction in the market and society [121]. In this light, the H2020 Multi-Annual Robotics Roadmap declared: "*Formal methods are needed to specify and synthesize robot control systems starting from rich models of the robot, of the environment, and of the task. [...] Tools like temporal logics and description logics have reached a good degree of maturity, and are a good candidate to create such methods.*"

Temporal logics such as Linear Temporal Logic (LTL) [108], Computation Tree Logic (CTL) [9], and Probabilistic Computation Tree Logic (PTCL) [58] provide formal high-level languages that can describe robot behaviors. These languages abstract and simplify the mission-specification task with respect to other approaches like the well-studied point-to-point navigation [80, 87, 89].

Originally resulting from the field of model checking and verification [7], LTL applied to robotic behavior planning provides a formalism to unambiguously specify the expected behavior. As such, an LTL specification can be used to describe the result of an expected behavior, while the way to achieve this result is automatically derived by the system. This means that as opposed to imperative point-to-point navigation specifications, an LTL specification is declarative and the steps to be taken by the robot are automatically computed (i.e., planned) by a planner. LTL is currently used due to the variety of existing planners that take LTL-based models as input [96, 57, 43]. Furthermore, CTL allows specifying properties that predicate on a branching sequence of assignments. For these reasons, LTL and CTL are being increasingly used by robotics experts for mission specification [97, 92, 56, 44, 96, 128, 43, 57, 137, 83].

PCTL is a language based on CTL that allows stating soft deadlines using probabilities and time bounds, which has been used for probabilistic model checking [119, 53, 85]. The community has proposed studies on probabilistic specifications

using the model checker PRISM and its capabilities of quantitative analysis [86] with which robotic applications have been modeled [27].

1.1.3 The Robot Operating System

This section provides an overview of key concepts of the Robot Operating System (ROS) [109]. We also describe aspects of the ROS community that we deem important to highlight.

1.1.3.1 The ROS Framework

According to the large-scale developer study presented in Paper A, ROS is the current *de facto* framework for robotics. It is already available for a huge variety of robots including, among others, ground mobile robots, static manipulators, autonomous vehicles, humanoid robots, unmanned aerial vehicles, and underwater robots.[12]

ROS is an open-source robotic framework that provides a message-based peer-to-peer communication infrastructure, called `roscore`, supporting the integration of independently developed software *packages*. ROS packages are the most atomic unit of build and the unit of the release of the ROS's ecosystem. The goal of packages is to promote reusability and modularity in ROS-based software development, also allowing writing robotic software that can be reused on different hardware platforms. There is no established rule on how to build ROS packages, but they tend to follow a "Goldilocks" principle, that is, to provide enough functionality to be useful, but not too much that the package is heavyweight and difficult to use by other software. Packages might contain different software artifacts that constitute a software module, including libraries, datasets, configuration files, third-party pieces of software, or ROS *nodes*. Nodes are independently developed software components that encapsulate processes that perform specific computational tasks, such as controlling actuators, running navigation algorithms, or processing images. In ROS, nodes are combined into a graph and communicate among them using *topics* or RPC *services*. Topics are named channels over which nodes exchange *messages* using a publish/subscribe paradigm. In turn, messages are a strictly-typed data structure that considers standard primitive types such as integer and boolean as well as more complex ROS-defined datatypes.[13] Differently, services are pairs of services that allow request/reply interactions. Most often, ROS applications depict graphs of nodes connected by topics, as stated by Estefo et al. [42].

The ROS framework is complemented by a set of tools for software development,[14] debugging,[15] simulation,[16] and visualization.[17] One of the main goals for this framework is to decouple robotic software from the hardware so as to make ROS-based applications robot-agnostic.

[12] `https://robots.ros.org`
[13] `http://wiki.ros.org/rosmsg`
[14] `http://wiki.ros.org/catkin/conceptual_overview`
[15] `http://wiki.ros.org/rqt`
[16] `http://gazebosim.org`
[17] `http://wiki.ros.org/rviz`

1.1.3.2 The ROS Community

One of the main keys to the success of ROS is the active community that supports the middleware [49, 42]. The development, maintenance, and variability management of ROS's main platform are centralized and performed by volunteers and employees of the Open Source Robotics Foundation, Inc. (OSRF).[18]

A free market is an essential goal of ROS. There are hundreds of ROS repositories hosted by companies, universities, and individuals around the world as GitHub repositories. They are indexed by the distribution directory of the `rosdistro` repository. The maintainers are responsible for managing and licensing the repositories, while in turn, they retain direct ownership and control over the code. As the ecosystem composed of distributed packages grows, the community needs to keep efficient communication channels. The most commonly used include ROS Answers[19]—a Q&A website formatted similar to Stack Overflow—, mailing lists, ROS Discourse[20]—forum in which users discuss topics regarding the future of ROS—, and GitHub.

ROS consumers include technically skilled end-users, such as robotic experts who develop individual ROS packages providing specific robotic functionalities as well as application experts, who develop complete robotic systems integrating a large variety of functional packages.

1.1.4 Software Architectures

Software architectures are currently key for the successful development of large and complex systems and play a vital role in ensuring the fulfillment of functional and non-functional requirements [116]. It is accepted that architecture-centric software development enhances the quality, modularity, and reusability [24] of systems. Through the history of software architectures, several definitions have been proposed [8]. The two seminal works of Garlan and Shaw [50] and Perry and Wolf [106] each provide definitions that we consider had an enormous impact on the software engineering community and adjust well to our research. Perry and Wolf [106] built the foundation for software architecture and a year later Garlan and Shaw [50] defined the term of software architectures as "*a higher-level abstraction of a software system that consists of a collection of computational components together with connectors that describe the interaction between these components*." A decade later, Bass et al. [8] wrote in their book "Software Architecture in Practice" that "*the software architecture of a system is the set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both*."

1.1.4.1 Robot Software Architectures

The earliest architectural approaches for autonomous robots in the late 1960s relied on the Sense-Plan-Act (SPA) paradigm [2, 81, 20]. This paradigm decomposes robotic control systems into three distinct subsystems: (i) the sensing system acquires sensory measures and updates an internal world model (*sense*); (ii) the planner uses the world model to generate a sequence of actions that would achieve a given goal (*plan*); and, (iii) the executor sends the actions to the actuator of the robot (*act*). These three subsystems are executed independently. One limitation of the SPA paradigm is the

[18] `https://www.openrobotics.org/company`
[19] `http://answers.ros.org`
[20] `https://discourse.ros.org`

centrality of the world model: it takes a long time to plan in any real-world domain and the world model becomes rapidly inaccurate in a dynamic environment. But most importantly, using this paradigm the plan is executed without actively using the sensors that created the model. The execution of a plan without involving sensing may cause dangerous behaviors in a dynamic world [81].

In the early 1980s, several new robot control architecture paradigms were proposed to overcome the drawbacks of the SPA paradigm. With the reactive paradigm [45], robotic systems could plan in a quick and reactive manner. The subsumption architecture of Brooks [20] was perhaps the most influential work of its time. The subsumption architecture consists of a layered collection of loosely interconnected modules that directly connect sensors to actuators. These modules are implemented as finite-state machines named *behaviors* (e.g., follow a target seen by the camera, or avoid an obstacle detected by the laser scanner). Behavior-based control architectures have proved to be successful for the design of relatively simple systems that mimic the behavior of simple insects and small rodents. However, it complicated composing behaviors to achieve long-range goals and optimizing the robot behavior [81].

Modern robot control architectures integrate the SPA deliberative approach with the behavior-based reactive approach in many different ways. One of the most popular architectural paradigms used to integrate those two approaches is layered architectures, in which each layer serves one purpose, encapsulates a certain functionality, and depends on the layer(s) above or below to complete a system's functionality [3]. A notable example is the CLARAty architecture [102] designed to support the heterogeneity of a number of NASA Mars Rovers.

During the last 20 years, the robotics research community has increasingly explored the synergies between robotics and software engineering to enhance its quality and reduce development costs. The pioneering work of Coste-Maniere et al. [31] sets the basis for this new trend by promoting architectural design as the central activity in robot system engineering. Architecture-centric robotics research and practice can be characterized by various architectural models that emerged over time such as: (i) Object-Oriented Architectures, enabling modularity [99], (ii) the Component-Based Software Engineering (CBSE) paradigm, supporting reusability [74], and (iii) Service-Oriented Architectures, exploiting dynamic composition of software [28].

Currently, robot control architectures are mostly designed according to the CBSE approach [3]. A possible explanation is that, as discussed previously (see Section 1.1.3), the Robot Operating System (ROS) [109] favors a software development approach that consists of designing software components, which implement common robotic functionalities and communicate by exchanging messages according to the publisher-subscriber paradigm. The strength of this approach is the possibility to develop a large variety of different control systems by composing in multiple ways reusable software building blocks. However, one of the most pressing associated challenges consists of selecting, integrating, and configuring a coherent set of components that provide the required functionality taking into account their mutual dependencies and architectural mismatches.

1.1.5 Variability Management

A core challenge affecting robotics software engineering is its need for *variability*, which is described by Galster et al. [47] as "*the ability of a software artifact to be changed in order to fit different contexts, environments, or purposes.*" A variety of

strategies and mechanisms to plan, design, and implement (i.e., to *manage*) variability has been proposed [130, 5, 11, 101, 33]. In the following, we introduce some of these strategies and mechanisms ranging from more ad hoc to more systematic variability management.

Clone & own is an ad hoc strategy to create variants of a product by copying and adapting existing variants to new requirements. This strategy is simple and cheap, but does not scale well with the number of variants [37, 13]. To overcome the scalability issues of this strategy, organizations could adopt *configuration mechanisms* to reduce redundancies. There are several techniques to be used for these mechanisms, and typically they generate configuration options (or calibration parameters), which are declared in a model. Organizations may also make use of clone-management frameworks [91]. These frameworks allow to integrate cloned variants into a configurable *platform* by following platform-oriented engineering methods such as software product line engineering [5, 130]. A clone-management framework also allows deriving individual variants from those previously cloned by configuring the platform. By implementing an integrated platform, an organization can also remove redundancies and integrate all variants instead of cloning them. Variability in a platform is represented by *features*, which represent a distinct and well-understood aspect of a system [12]. To structure and keep a common understanding of features, organizations may create a *feature model* [10, 34, 101]—tree-like structures organizing features in a hierarchy, together with constraints among the features.

1.2 Goals of the Thesis

As stated in the introduction, the overall goal of this thesis is to investigate practices and challenges in service robotics software engineering to provide guidance for practitioners and researchers as well as to engineer solutions for some of the challenges we identified. To this end, we formulated seven research questions that map to our goals as follows. Research questions RQ1 (including RQ1.1 and RQ1.2) and RQ3.1 are formulated to cover the first of our research subgoals (see the introduction), **G1**: *to provide empirical insights of the current state-of-the-art and -practice of service robotics software engineering that lead to establishing effective practices, methods, and tools for engineering service robotics applications*. On the other hand, RQ2, RQ3.2, and RQ4 are framed in the context of the second subgoal, **G2**: *to identify, explore, and engineer solutions for some of the practical problems in the service robotics domain from our findings in **G1***.

RQ1. What are the current software engineering practices for developing service robotic applications? (Papers A and B)

RQ1.1. What are the current software engineering practices for developing service robotic applications and how are they shaped by the distinguishing characteristics of the domain and its challenges? (Paper A)

RQ1.2. What are the most recurrent drivers of variability in the development of service robotic applications, what challenges do they pose, and what practices do practitioners apply to overcome them? (Paper B)

The data we collected to answer RQ1 may be exploited to establish methods and tools for developing and managing robotic applications. Both Paper A and Paper B report several challenges and limitations in the service robotics software engineering domain, focusing on applications in industry. Paper A follows a broad approach to

assess applied practices, distinguishing characteristics, and challenges of the domain while Paper B studies the same domain focusing on the impact of variability and its management. Based on the limitations identified through the study of Papers A and B, we decided to focus on and study three issues that we considered fundamental to improve the development pace of service robotics software engineering. These topics are software architectures, robot mission specification, and planning of robotic applications under uncertainty. We formulate three research questions for these topics.

RQ2. How can software architectures help to build decentralized and self-adaptive applications for heterogeneous multi-robot teams? (Papers D, G, H)

The first research question of our engineered solutions arises from the identified lack of common software architectures in service robotics software engineering. Software architectures are considered necessary to structure and build robotic applications in a systematic manner [3], enhancing their quality, modularity, and reusability [24]. Still, the H2020 Multi-Annual Robotics Roadmap (MAR) [121] stated that current robotics software development processes lack architectural models. We consider it crucial to understand how software architectures support robot development and to what extent software architectures simplify software development, integration, and deployment within a specific domain. We investigate the state-of-the-art on the topic in Paper D and engineer a solution, which is put to test in Paper G and Paper H.

RQ3. How can tools and mechanisms such as Domain-Specific Languages and Integrated Development Environments support operators specifying the intended behaviors of robotic systems? (Papers C, E, G, H)

RQ3.1. What Domain-Specific Languages and Integrated Development Environments exist to support the specification of mission and behaviors of robotic systems in end-user domains? (Paper C)

RQ3.2. How to support operators in the specification of complex missions for teams of heterogeneous robots combining expressiveness, simplicity, and rigorousness? (Papers E, G, H)

The MAR foresaw a near future where service robots are deployed in a variety of social and industrial environments, which, as discussed in the introduction, is happening nowadays. As service robots operating in hospitals, hotels, or warehouses become a reality, the need for mission specification mechanisms and strategies to operate them grows in importance. Paper C provides an overview of Integrated Development Environments and Domain-Specific Languages (DSLs) in robotics and Paper E proposes a language and integrated tool for operators. In Papers G and H, we describe the integration of our DSL into a framework and its testing in real-world scenarios.

RQ4. What are the applicable strategies to manage heterogeneous robotic applications with only partial knowledge of dynamic environments? (Paper F)

Robots must be able to operate in partially constrained, dynamic environments with the presence of uncertainty [121] (e.g., has the environment changed so the computed plan is not valid anymore? Is the goal of the mission still achievable or does an environmental event impede it?). Therefore, we build upon an existing planning component and instill it with the capability of enabling teams of heterogeneous robots to operate in environments with the presence of uncertainty.

1.3 Research Methodology

Robotics involves researchers, practitioners, and developers from several disciplines (e.g., mechatronics, computer vision, control theory, software engineering) who need to collaborate in the process of developing complex, safety-critical systems. Due to the crucial role of human behavior in this process and to the overall lack of empirically captured insights in service robotics software engineering, the domain needs to be studied with systematic and appropriate empirical research methods. Based on the knowledge and insights obtained through empirical methods that identify the industry's problems and needs, candidate solutions can be engineered and evaluated.

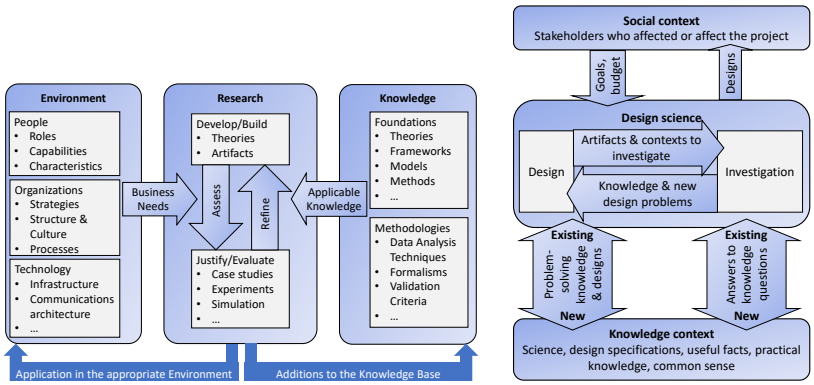
With this aim, we conducted a combination of *knowledge-seeking* and *solution-seeking* studies, as described in the studies of Stol et al. [124, 125], where it is explained that software engineering studies can be roughly classified into these two categories. The former category aims to learn about the world around us. In the specific context of service robotics software engineering that includes practices, challenges, trends, and artifacts from the robotic software development process. The studies of the latter category observe a certain technical or practical problem and *engineer* a solution to address it.

To help overcome the limitations and challenges that such a vibrant and complex domain poses, we conducted three knowledge-seeking studies (i.e., Papers A–C). The knowledge and insights from those studies permitted us to identify technical and practical problems in the domain. We then conducted five solution-seeking studies (Papers D–H) to overcome the identified problems by engineering solutions that meet industrial needs.

The overall methodology followed in this thesis to reach the combination of knowledge-seeking and solution-seeking studies is inspired by the design science research paradigm. Design science is a research paradigm whose overall goal is to solve real-world problems [41]. According to Wieringa [136], design science consists of the iterative design of an artifact that improves something for stakeholders within a project or context and its validation and evaluation through empirical investigation. In this thesis, we refer to Van Aken's definition for research paradigm: "*the combination of research questions asked, the research methodologies allowed to answer them, and the nature of the pursued research products*" [129].

The work of Hevner et al. [63] provides an overview of the design science paradigm in the context of information systems and illustrates it by proposing a conceptual framework. The framework depicts three major units (see Fig. 1.3a). The first of them is the *environment* (i.e., the problem space containing the phenomena of interest), and includes people, organizations, and technology. The second one, namely the *knowledge base* is composed of foundations (e.g., theories, frameworks, models) and methodologies. The third unit, the *research*, acquires the business needs from the environment and the applicable knowledge from the knowledge base to iteratively build and evaluate practically useful artifacts.

Wieringa [136] conceived another framework, largely based on that of Hevner et al. [63], but considering software engineering practices. A major difference between these two frameworks is that Wieringa's separates design (i.e., designing an artifact to improve a problem context) and investigation (i.e., answering knowledge questions about an artifact in a given context). Wieringa's framework consist of several components (see Fig. 1.3b), explained in the following. The *social context* comprehends the stakeholders (e.g., operators, maintainers, instructors, developers,



(a) Adapted from Hevner et al. [63]. (b) Adapted from Wieringa [136].

Figure 1.3: Conceptual frameworks for design science.

researchers) who may affect or be affected by the project in context. The *knowledge context* contains the body-of-knowledge applicable to a project, which in turn includes theories from different science and engineering disciplines, specifications of currently known designs, models, and frameworks, as well as reported experience of researchers in earlier design science projects. Knowledge is typically divided into prior knowledge (i.e., available before the project) and posterior knowledge (i.e., produced as a result of the project). These types of knowledge may be used in the framework to answer knowledge questions and may be extended by producing new designs.

In our knowledge-seeking studies, we extracted the state-of-the-art- and -practice in the studied domain, that is, service robotics software engineering. This helped us understand the environment and knowledge components of the framework proposed by Hevner et al. [63], or as put by Wieringa [136], the social and knowledge contexts of the domain. In other words, we learned about existing business needs and applicable existing knowledge so we were able to formulate new goals for our research. By means of the conducted solution-seeking studies, we engineered solutions through building and evaluating (Hevner et al. [63]) or designing and investigating (Wieringa [136]) iterations.

Figure 1.4 provides an overview of the methodology that we followed for this thesis, labeling activities and contributions, which are then referenced to the appended papers. The figure depicts activities associated with design science as columns, namely problem/need identification as well as design, development, and investigation of solutions. It is important to remark that design science, as opposed to other research methods like case studies focuses on the design of an actual artifact instead of understanding a situation in its context without actively changing it. Hence, the research conducted during our knowledge-seeking studies (i.e., Papers A–C) solely constitutes the problem or need identification processes we followed (e.g., identifying the need for architectural solutions in the robotics industry).

Table 1.1 lists the research methods used in this thesis for collecting and analyzing empirical data—either qualitative, quantitative or both—in this thesis. These methods will be further discussed accordingly in Sections 1.3.1 and 1.3.2.

The research presented in this thesis has been conducted in the context of the

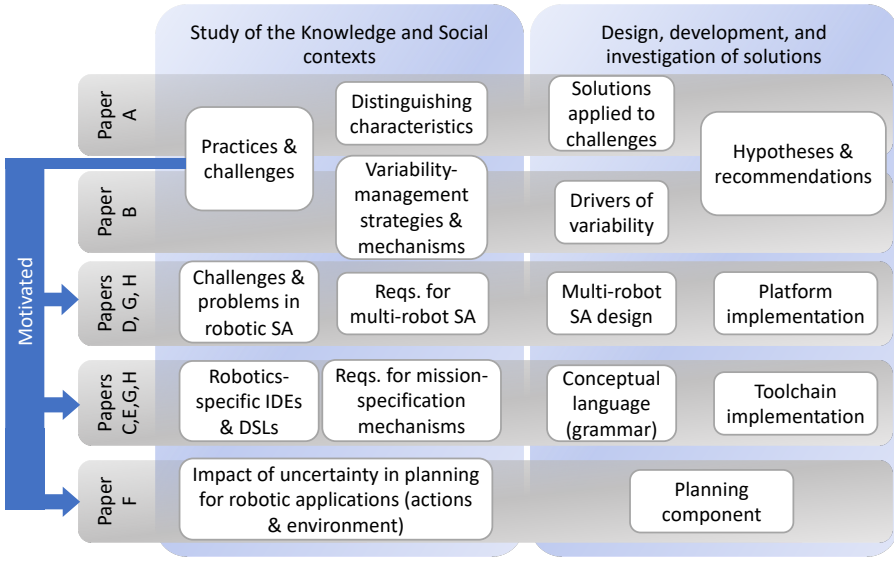


Figure 1.4: The research methodology followed in this thesis.
Note: Software Architecture(s) (SA), Requirements (Reqs.).

European project Co4Robots.[21] The project consortium was formed of four academic and two industrial partners, which are described below. Working with these two companies allowed us to design and investigate our engineered solutions within an industrial frame and to ensure that they meet business needs.

PAL Robotics[a] (Spain) is a medium-sized robotics company that mainly manufactures humanoid robots. The company provides robotic solutions to both research institutions and companies, focusing on the areas of research, logistics, retail, and social robotics. At the company, the humanoid robots are considered mainly *research platforms* while TIAGo base[b] and StockBot[c] are considered *professional service robots*.

The Bosch Center for Artificial Intelligence[d] (Germany) is a medium-sized company focused on developing innovative AI technologies for Bosch. The Dynamic Multi-Agent Planning division is interested in optimizing the goal achievement of multi-agent systems safely while enabling operators to specify goals and requirements directly and unambiguously. For that task they mainly use *research prototype robots* such as the Intelligent Transport Assistant (ITA).

[a] http://pal-robotics.com/
[b] https://pal-robotics.com/robots/tiago-base/
[c] https://pal-robotics.com/robots/stockbot/
[d] https://www.bosch-ai.com/

[21] http://www.co4robots.eu/

Table 1.1: Research methods used for empirical data.

Paper	Research method	Data sources
A	Mixed methods	Online questionnaire (156 answers) 28 semi-structured interviews (16 companies)
B	Case study	11 interviews (2 companies)
B	Systematic literature review	28 papers
D	Focus groups	3 focus groups with a total of 21 participants
D	Online questionnaires	2 online questionnaires (6 responses to each of them)
E	Focus groups	4 focus groups with a total of 21 participants
E	User studies	2 user studies, with 9 and 6 participants
F	Focus groups	2 focus groups with a total of 21 participants

1.3.1 Knowledge-Seeking Studies

For the study of Paper A, we interviewed and surveyed practitioners from many robotics companies all around the world to collect qualitative and quantitative data by using *mixed methods research*. It is a complex research method that uses data collection and analysis techniques for both qualitative and quantitative data. It emerged due to the potential weaknesses of using separate research methods with the idea of compensating for those using other methods so as to apply their strengths [32, 39]. This method permits researchers to obtain a more extensive picture of a studied phenomenon however at the cost of being more challenging than other methods due to the need to extensively collect, analyze, and triangulate the required data [39, 68]. In paper A, we conducted mixed methods research using a sequential exploratory design and collecting, analyzing, and triangulating data from an online questionnaire and semi-structured interviews [32, 39]. We started the research with an exploratory study consisting of four semi-structured interviews with robotics practitioners. Based on what we learned from them we designed an interview guide and, in parallel, a survey questionnaire. After the exploratory study, we conducted another 14 interviews with an updated version of the interview guide and published the designed online survey. In summary, we interviewed 18 practitioners from 15 different companies, who work in 9 different countries. We collected 156 answers to our online survey, from which we could extract 94 email addresses based on the metrics of the respondents' demographics, which we used to track the company for which the respondents work. Out of the total, 22 correspond to academic practitioners from 16 different universities from 10 different countries. The rest of the email addresses (72) correspond to industrial practitioners working for 58 different companies from 20 different countries, as depicted in Fig. 1.5.

In Paper B, we conducted a stage-based methodology that combines data from three different sources through triangulation. In the first stage, we gathered our experiences from different projects—our experience stems from various EU, academic, and industrial robotics software engineering projects—and enriched them with two interviews with robotics experts. In the second stage, after identifying the main topics we wanted to explore in detail, we designed a *multiple-case study* [39] and contacted and interviewed nine robotics experts working for two different companies with headquarters in two countries (i.e., Spain and Denmark). The third stage consists of a Systematic Literature Review. In this stage we analyzed 178 papers from which, based on the inclusion

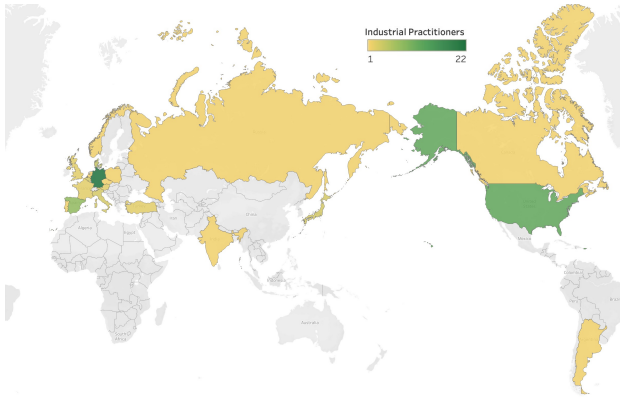


Figure 1.5: Map of responses from industrial practitioners to our online questionnaire as tracked based on the respondents' demographics.

and exclusion criteria outlined in Paper B, we selected 28 and extracted data from them.

Paper C consists of an overview of tools and mechanisms available for researchers and engineers to program robots. With this aim, we investigated 15 robotics-specific Integrated Development Environments (IDEs) and discussed details on target users, languages supported, features that go beyond a general IDE, and how they support roboticists. The main focus of the study is however in Domain-Specific Languages (DSLs) for mission specification in robotics, for which we investigated a total of 15 DSLs. We discuss the application of those languages, their tool implementation, their notation, and how they support the operator during mission specification.

1.3.2 Solution-Seeking Studies

We conducted five solution-seeking studies (Papers D–H) with the aim of engineering solutions for problems that were identified during the knowledge-seeking studies. In particular, we investigated and engineered software architectures (Papers D, G, H), mission-specification languages (Papers C, E, G, H), and planning components (Paper F). To engineer our solutions we conducted iterative steps of design, development, and investigation, the latter most often through experimentation and empirical validation.

Paper D introduces SERA, a software architecture conceived to support decentralized multi-robot applications. The research conducted in Paper D stems from the necessity of supporting the building of robotic applications in a systematic manner while observing specific features (e.g., decentralization, heterogeneity, self-configuration). The business needs behind this necessity were identified and corroborated through the study of Papers A and B and the MAR [121] as well as through focus groups [77] with the Co4Robots's consortium. Concretely, as stated in the MAR: "*Usually there are no system development processes (highlighted by a lack of overall architectural models and methods). This results in the need for craftsmanship in building robotic systems instead of following established engineering processes*." The development of SERA comprehends an iterative and continuous process of design, development, and investigation that lasted several months and was further updated in the following three years. To identify the requirements for such an architecture we leveraged data collected from

focus groups, demonstrations, and informal surveys (Table 1.1). This was complemented by identifying current problems and challenges in robot software architectures in the literature, especially from the systematic mapping study conducted by Ahmad and Babar [3]. Our software architecture's main features, that is, following a 3-layer hierarchy and a component-based approach as well as supporting self-adaptation, are based on those of the well-known architecture by Kramer and Magee [82]. The investigation activity required several iterations of validation through experimentation—in both simulation and real-world scenarios—and collecting empirical data from focus groups and demonstrations. The evolution of SERA lasted for the whole duration of the Co4Robots project (i.e., 38 months), from its requirements elicitation, theoretical foundation, software implementation, and realization as a distributed framework. The experimentation aspect of the validation and the evolution of SERA through the years are documented in Papers D, G, and H.

In Paper E we introduce PROMISE, a Domain-Specific Language (DSL) developed to support operators in the specification of missions for heterogeneous multi-robot teams. We identified the need for rigorous and expressive yet user-friendly mission specification mechanisms through focus groups with the Co4Robots's consortium. This was later corroborated and further motivated by conducting Papers A–C and carefully reading the MAR [121]. The latter states: "*robots will need to become intuitively integrated with human operators. Systems will be designed to complement and act as an aid to a human mission expert*." To conceive our language, we studied the knowledge context and received inspiration from the formalism of behavior trees. As detailed in Paper E, the design and development of PROMISE consisted of several iterations. The ability of PROMISE and its toolchain implementation to be used to specify and execute a variety of missions by operators without experience in robotics was validated empirically by conducting two user studies with nine and six participants, respectively. PROMISE was also validated through experimentation in both simulation and real-world scenarios, as detailed in Papers E, G, and H.

Paper F proposes a planning solution that decomposes a team of robots into subclasses, considers missions given in temporal logic, and at the same time works when only partial knowledge of the environment is available. We identified the necessity of supporting planning for robotic applications operating in environments with the presence of uncertainty for paper F in a similar manner to Papers D and E. That is, by conducting two focus group sessions with the Co4Robots's consortium in the first months of the project, conducting the research for Paper A, and studying the MAR [121]. As put by the latter: "*In many real applications there will be a task balance between the capture of new knowledge about an environment and the execution of the task at hand with incomplete or uncertain data. The development of decision support mechanisms that are able to manage this balance will be critical to a number of application domains*." Our first step was to explore existing problem-solving knowledge and designs from the knowledge context. We selected an appropriate planner already available in the literature [127], which we used to validate our solution. To validate our study and the conceived theory, we implemented MAPmAKER upon Tumova and Dimarogonas's planner as a proof-of-concept standalone application using MATLAB.[22] The algorithms were evaluated through experimentation in different simulated scenarios with different configurations.

[22] `https://se.mathworks.com/products/matlab.html`

1.4 Summary of Contributions

This section details our contributions as a summary of the appended papers to this thesis. In the following, we describe each paper's research motivation, goal, and how they contribute to answering the research questions introduced in Section 1.2. We refer the reader to the individual papers appended in Chapters 2–9 for a detailed discussion of the related work.

1.4.1 Robotics Software Engineering (Paper A)

Traditionally, the most successful field in robotics has been factory automation, where robots have built-in proprietary controllers for repetitive tasks that allow a simple programming style. In this context, software engineering has been considered an auxiliary concern for robotic system construction. The focus of research on robotic software is often on inventing new algorithms (e.g., for sensor fusion, motion planning, deliberative control) instead of on structuring robotic software [23]. Despite more studies are being conducted in the very last years, there is little evidence about the concrete software development practices, methodologies, and tools applied in software engineering for service robotics. The study reported in Paper A is motivated by this lack of empirical knowledge.

To build effective and safe robots, it is crucial to provide guidance to better support practitioners, increasing the adoption of software engineering best practices. Our long-term goal in Paper A was to establish effective software engineering practices for a substantial lifting of the state-of-practice for engineering service robotics and provide guidance for practitioners. We also aim to help the research community to scope future research that meets industrial needs. Paper A constitutes a comprehensive assessment of the state-of-practice in service robotics software engineering following the methodology detailed in Section 1.3.1.

Contributions. The main contributions of Paper A include the findings from our interview data and results from our online survey, which combine qualitative and quantitative data. We first report on roboticists' software development practices, that is, applied development activities, processes, paradigms, software languages and frameworks, as well as quality assurance and reuse practices. The study also provides information on the used tools for the practices described above. Secondly, the study investigates the distinguishing features of service robotics software engineering, especially with respect to other cyber-physical domains (e.g., automotive, avionics). Third, we surveyed practitioners to determine the specific software engineering challenges in service robotics, their impact, and how they are addressed. Finally, we synthesize the collected data through triangularization and propose three research themes that pose important software engineering challenges and limitations in service robotics according to the observations we made throughout the paper. Each research theme contains a summary of its associated observations, hypotheses we formulate from them to explain the phenomena, and our proposed actions for researchers and engineers to improve development processes in said themes.

1.4.2 Software Variability in Service Robotics (Paper B)

A core challenge affecting service robotics software engineering is *variability*—the ability of a software artifact to be changed in order to fit different contexts, envi-

ronments, or purposes [47]. Robotic software needs to account for a diversity of hardware, operating environments, and customer demands, without being redeveloped from scratch. Similar to other domains that need to cope with variability, such as automotive, avionics, telecommunication, and industrial automation [13], variability in robotics arises from dedicated *drivers of variability*. These drivers stem from the diversity of hardware, operating environments, and missions the robots may perform. As such, robot developers need to manage variability in their software using so-called variability mechanisms, which are implementation techniques to realize variation points. However, while drivers of variability and variability mechanisms are reasonably well understood in other domains, that is not the case for the vibrant domain of (autonomous) service robots. If not managed properly, variability in complex systems may lead to failures, including drops of performance or crashing of the system [123, 135], or as pinpointed by the MAR: "*Robots that operate in environments with high levels of variability, both in terms of objects and in terms of dynamics, will inevitably display lower levels of dependability for a given technology level than robots working in structured and near static environments*" [121].

Our long-term goal in Paper B was to improve the methods and tools for managing variability in the robotics domain. Variability in robotics is mainly described by drivers not seen in other domains and is managed in ways that lack behind the state-of-the-art in systematic variability management. However, to improve the situation, we first need to improve our empirical understanding of variability in service robots. With this aim, we investigated the drivers, practices, methods, and challenges of variability from companies that provide service robotic applications. We analyze the state-of-practice and the state-of-the-art—the former via an experience report and eleven semi-structured interviews with industrial experts from two service robotics companies; the latter via a Systematic Literature Review (SLR).

Contributions. In Paper B, we identified the characteristics of the drivers of variability that affect our subject companies, the practices applied to manage them, and the challenges that variability and its management pose. We discuss their impact on the studied companies' development processes. In summary, the main contributions of Paper B are

- Qualitative empirical data about drivers of variability, management mechanisms and strategies, as well as challenges.
- An SLR on variability management in service robotics.
- A comparison between the state-of-the-art and the state-of-practice in on the topic of variability drivers, management, and challenges in service robotics.

Similar to what we did for Paper A, another contribution of Paper B is the grouping of key observations into themes that we deemed deserve further investigation. Based on that grouping, we formulated hypotheses from the observations and propose actions for researchers and practitioners.

1.4.3 Multi-Robot Software Architecture (Papers D, G, H)

Service robots support humans by performing specialized tasks; however, the complexity of certain tasks may require more than one robot for their achievement. Furthermore, service robots are often required to operate in highly dynamic environments, possibly populated by humans [121]. In certain situations, robots might even be required to

collaborate or assist humans. To enable robots to operate in such scenarios, today's robotic applications demand architectures that support inter-robot collaboration and human-robot interaction along with self-adaptation.

In the last years, research in the field of self-adaptation has grown significantly [94] and many architectures for self-adaptive systems have been proposed [35, 132, 94]. The body-of-knowledge generated by those studies has been applied to the robotics community [3], which derived in robot software architectures supporting *self-** features (i.e., self-configuration, self-adaptation and self-healing, self-monitoring, and self-tuning) [82, 27, 94, 40, 51, 69, 18]. Nevertheless, most of the existing robot architectures that support self-* features typically do not support multi-robot and robot-human decentralized collaboration or have not been tested in real-world scenarios.

In the context of robotic coordination, decentralization has been deemed important since it: (i) allows managing large teams of robots, (ii) facilitates—through loose coupling—the addition, substitution, and removal of robots, (iii) minimizes the cost of changes since the responsibility of every action is within every single robot that observes its environment and acts on events, (iv) facilitates data-centric workflows since data are passed directly to where they are required, and (v) increases robustness and decreases system vulnerability (e.g., reducing single points of failure) [138].

The main goal of the research conducted in Paper D (and partially in Papers G and H) is to support the systematic building of software applications for heterogeneous multi-robot teams that operate and collaborate in a decentralized and self-adaptive manner. With this aim, we designed and implemented a software architecture by following the methodology detailed in Section 1.3.2. In Paper D, we explain the design and first implementation of our architecture and its platform along with their first evaluation steps. Paper G describes how SERA and its platform are applied to build a robotic application that supports robot to robot and human-robot collaboration. Finally, Paper H details the realization of SERA and its platform as a framework that supports a number of features to support the building and operation of robotic applications.

Contributions. The main contribution of Paper D is SERA, a Self-adaptive dEcentralized Robotic Architecture. SERA is defined as a three-layered, decentralized architecture, that follows a component-based approach. Our architecture promotes the development of applications that embed self-* features. SERA is integrated into a software platform that integrates various software components developed by other partners from Co4Robots (e.g., self-localization tools, motion planning and control algorithms) through suitable glue code. More specifically, the glue code represents the logic concerning the interfaces among the architecture's components. We showed the feasibility of implementing SERA by realizing a prototype using ROS.

SERA enables integrating the solutions proposed in Papers E and F as components of its architecture (see Figure 1.6). In particular, PROMISE (Paper E) implements the components of the architecture's *Central Station* unit. The interpreter of PROMISE's toolchain implements the component *Local mission manager*. Then, MAPmAKER is a planning solution that could be integrated as the component *Adaptation manager*. For Papers D, G, and H, robotic capabilities (i.e., perception, navigation, and manipulation) are implemented in a modular way by a developer. They are referred as to *components* in Papers D and H and as to *skills* in Paper H.

Paper H describes the last version of SERA and its realization as a distributed framework that allows for automatically composing and adapting heterogeneous multi-robot behavior from a set of individual robot capabilities. Every Paper H's skill is modeled as a labeled Markov Decision Process (MDP), i.e., a well-studied model of sequential

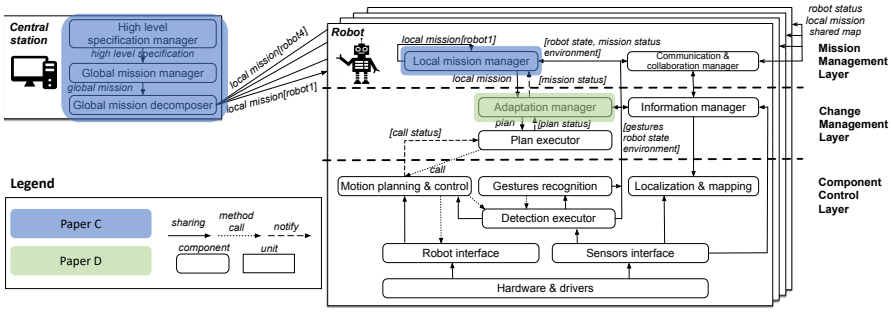


Figure 1.6: SERA architecture integrating Paper E and (possibly) F.

decision making under uncertainty that has been applied in a variety of significant real-world settings [134]. The framework described in Paper H also integrates mission specification capabilities consisting of task-specific patterns, which may be exploited by a domain expert either programmatically, using PsALM [98, 95] or using PROMISE (explained shortly in Section 1.4.4). The mission specification patterns build upon the existing catalog of patterns used by PROMISE (as explained in Section 1.4.4). In this framework, both the skills from each robot that will take part in the mission and its specification are combined to a task model whose formulation is distributed so each robot only models a local subset of the complete system. To allow the execution of the mission specification, Paper H's framework provides task planning capabilities that require an input in form of a *Linear Temporal Logic* (LTL) specification. LTL is used to guarantee that only skills and actions within the skill MDPs are selected that conform with the given specification. In other words, any sequence of propositions resulting from the actions of the robots and the modeled dynamics of the environment should satisfy the given task formula. The use of LTL as the formalism to express tasks and the combination with online auctions [114] for dynamic task allocation allows for an automated decomposition and meaningful collaboration between the different robots. Whenever new online observations are obtained, the re-allocation of tasks ensures that the team always follows the specified reactions and incorporates possible additional tasks.

1.4.4 Mission-Specification DSL (Paper C, E, H)

Imagine a social or industrial facility such as a hotel, an office, a hospital, or a warehouse where a team of robots has been deployed to provide services and accomplish everyday tasks. Such robots may help humans by performing monotonous, menial, or dangerous tasks. In those contexts, service robots with various capabilities need to be commanded or coordinated by an operator for them to achieve such tasks. To meet these business needs, we consider it crucial to conceive mission specification tools, mechanisms, and methods for operators to easily command missions to teams of robots.

As explained in Section 1.3.2, the MAR [121] has foreseen the need for human (possibly non-technically skilled) operators to specify missions for robots that are now part of their everyday jobs.[23] This calls for tools, mechanisms, and methods of mission specification to be user-friendly yet rigorous. Domain-Specific Languages

[23] https://spectrum.ieee.org/automaton/robotics/medical-robots/autonomous-robots-are-helping-kill-coronavirus-in-hospitals

(DSLs) have gained traction in the last years because they aid domain experts to easily define their concerns by providing robotics-tailored tools, mechanisms, and methods [103, 104, 118]. In Paper C, we conduct a study on mission specification languages currently used in robotics, provide an overview, and discuss their main characteristics including style, notation, and main features. The study's main focus is on languages available online for end-user domains. Paper E provides another study of the state-of-the-art on mission specification languages, this one focusing on research studies. The goal of Paper E is to provide a language, toolchain, and implementation that support operators in mission specification combining expressiveness, simplicity, and rigorousness. Concretely, Paper E presents PROMISE (simPle RObot MIssion SpEcification), a DSL that (i) provides a user-friendly syntax while having well-defined semantics; (ii) enables a rigorous and precise specification required for the use of planners, analysis tools, simulators or other modules; (iii) permits the specification of complex missions by providing executable and combinable tasks and operators; and (iv) is platform-independent and highly customizable.

Papers G and H convey the usage of the results of Paper E in four real-world scenarios and their integration into the framework described in Section 1.4.3.

Contributions. The main contribution from Paper C is a comprehensive study of the state-of-the-art of robotics-specific IDEs and DSLs, which mainly focuses on end-user domains. Paper E presents PROMISE (simPle RObot MIssion SpEcification), a DSL that provides a graphical and textual interfaces (each of which is supported by a dedicated editor) to support operators in mission specification. Our language builds upon a catalog of mission-specification LTL-based patterns [97, 95]. Those patterns were proposed by Menghi et al. to perform mission specification in temporal logics from recurrent mission specification problems, ensuring the correctness of the semantics while at the same time raising the level of abstraction with respect to imperative specification languages or general-purpose languages. We refer to such patterns as *tasks*. PROMISE allows the composition of those tasks into more complex missions using *operators*, which were inspired by those from the behavior trees formalism [29].

PROMISE was developed as a standalone language that can be integrated into a variety of frameworks. In its current implementation, PROMISE is integrated into a toolchain that:

- Allows the decomposition of global missions into robot-specific local missions.
- Provides a transformation engine that automatically generates an intermediate language, which encodes a number of LTL-based formulae.
- Enables the mission parsing and execution of the mission by an interpreter.

Furthermore, PROMISE was first implemented into the platform of SERA—as described in Papers D and G—and was later exploited by the framework described in Paper H (see Section 1.4.4). Specifically, it implemented SERA's *High-level specification manager*, *Global mission manager*, and *Global mission decomposer* components. The interpreter integrated into PROMISE's framework is implemented as the *Local mission manager* (see Figure 1.6). Being integrated into the platform and framework allows our language to exploit robotic functionalities including planning, self-localization, and motion planning to compose complex missions and thus being evaluated in several scenarios.

1.4.5 Planning under Uncertainty (Paper F)

Robotic applications usually rely on planners that given a robot or a team of robots, compute plans that specify how the robot(s) can fulfill their missions. Current robotic applications require planners for automated planning that address two main challenges: (i) the planning algorithm should work when (only) partial knowledge about the system—including the robots and their working environment—is present; and (ii) the planning problem should incorporate methods and tools that help to reduce the planning overhead, allowing replanning during execution.

The main goal of Paper F is to provide a decentralized planning solution that considers missions given in temporal logic and at the same time works when only partial knowledge of the environment is available.

We define the concept of partial robot model in Paper F in terms of three aspects of robotic missions that may be affected by uncertainty, concretely:

- *Partial knowledge about the execution of the actions.* MAPmAKER's representation of the environment is a grid composed of cells. To represent the impossibility of transitioning between cells not all movements are allowed. Knowledge of transitioning between two cells may be partial if, for instance, the state of a door between two rooms is uncertain.

- *Unknown service provisioning.* Robots provide services (e.g., collect an object) in specific locations of the environment (i.e., cells). The provision of specific services might be uncertain under certain contexts if, for instance, there is partial knowledge of the whereabouts of the object targeted to be collected.

- *Unknown meeting capabilities.* Some services are complex enough to require the collaboration of two or more robots for their achievement.[24] The meeting possibilities of robotic applications may be uncertain if only partial knowledge of the meeting location is known. For instance, two human operators may be working at the station where the robots are intended to meet and collaborate.

Contributions. Paper F's main contribution is MAPmAKER (Multi-robot plAnner for PArtially Known EnviRonments), a novel decentralized planner for partially known robotic applications. MAPmAKER workflow starts by decomposing a given robotic application into subclasses—dictated by the mission to be achieved by each robot—and computes plans for each robot within the subclasses in a decentralized manner. Plans may be definitive or possible, i.e., definitive plans ensure the satisfaction of each robot's mission, while possible plans may satisfy the local mission due to some unknown information in the robotic application. MAPmAKER then chooses among the computed plans the one to be executed based on defined policies—e.g., always opt for shorter plans even if they contain uncertainty, avoid possible plans. Partial knowledge is handled by calling twice a classical planning algorithm—e.g., a robot must replan if the path of its possible plan goes through a closed door. MAPmAKER was designed to be integrated into SERA's architecture to provide its platform with a planning solution as an implementation of the *Adaptation manager* component (see Figure 1.6).

[24] `https://www.youtube.com/watch?v=q7dMLawf0y0`

1.5 Discussion

In the previous sections, we have introduced the seven appended papers and their main goals and contributions. In the remainder of this section, we discuss and synthesize the answers to the research questions formulated for this thesis.

RQ1. What are the current software engineering practices for developing service robotic applications?

Papers A and B employ empirical methods to study the state-of-the-art and -practice of service robotics software engineering. The former investigates—pursuing a broad approach—the applied practices to develop software and studies the impact of the main characteristics and challenges of the domain. The latter studies the same domain with a focus on software variability. Concretely, Paper B investigates the main drivers of variability in the development of service robotic software applications. To answer RQ1, Paper B also investigates the practices (i.e., strategies, methods, mechanisms, and tools) commonly used to manage the variability stemming from such drivers as well as existing challenges.

RQ1.1. What are the current software engineering practices for developing service robotic applications and how are they shaped by the distinguishing characteristics of the domain and its challenges?

In Paper A, we collected information from practitioners working in service robotics via interviews and an online questionnaire to understand the state-of-the-practice of that particular domain. Service robotics software engineering is a special software engineering domain, which is concerned with all the aspects of software development for robots [21]. To answer RQ1.1, we empirically studied three questions: (i) What practices are applied in software engineering for service robots? (ii) How is service robotics software engineering perceived to be different from software engineering in other domains? And (iii) what challenges do practitioners face related to software engineering for service robots?

The first question comprehends the topics of activities, development paradigms, development processes, software languages and frameworks, quality assurance, reuse practices, and used tools. Data shows that the activity most commonly performed by roboticists is implementation (99% of the survey respondents perform this activity always or very often). Surprisingly, real-world experimentation is performed more often than testing and simulation with 88% and 79%, respectively. These activities are usually performed using frameworks for agile methodologies (e.g., Scrum). To perform such activities, roboticists usually rely on software frameworks, especially the Robot Operating System (ROS) [109]. Aligned with the widespread use of ROS, the most used programming languages are C++ and Python—most libraries in ROS are written in either of those languages—, which are used always or very often by 89% and 83% of the respondents, respectively. Also, the most applied paradigms are object-oriented programming (OOP) and component-based software engineering (CBSE) with 92% and 65% of the respondents, respectively. The usage of OOP might be explained by the widespread usage of C++ and Python while the usage of CBSE could be a consequence of the popularity of ROS, whose way of structuring data into modules enforces this paradigm. The most applied quality assurance techniques are code reviews and testing, integration testing being the most repeated testing technique

(65% of the participants always or very often apply it). The common usage of integration testing might be caused by the popularity of ROS, which, as said before, enforces the CBSE paradigm for developing software and therefore requires the integration of several software modules to build a robotic application. Although one of the most commonly named characteristics of robotics software engineering is its cyber-physical nature (explained shortly), performance testing is rarely performed (only 36% of the respondents performed it always or very often). Surprisingly, simulation (82.1%) and runtime monitoring (75%) are almost equally important sources of testing data. This might mean that robotics practitioners perform almost as much simulation as real-world testing. The most reused types of artifacts by roboticists according to our interviewees and respondents are source code and libraries: 82% and 80% of the respondents stated that they always or very often reuse them. However, documentation is not reused very often (47%), which might be linked to some of the main roadblocks for reusing artifacts and challenges we identified from our data. The main roadblocks for reusing artifacts are mainly technical problems (78.2%) and lack of documentation (46.8%). Respondents stated using specialized tools for the most common activities (i.e., implementation, real-world experimentation) and generic tools for the least common ones (i.e., requirements engineering, automatic code generation).

For the second question, we studied the perception of our interviewees and respondents of service robotics software engineering, emphasizing the distinguishing characteristics of the domain. The most repeated characteristic was the robots' very cyber-physical nature, which was mentioned by 64 survey respondents out of the total of 156. Robotic systems integrate hardware and software components to be able to operate in and interact with the environment. The development, integration, and customization of hardware, software, and environmental components give rise to the significant complexity of service robotic systems. The management of such complexity calls for systematic software engineering practices, methods, and tools. The second most repeated characteristic—mentioned by 24 respondents—was the heterogeneity of disciplines that are required to develop a full-working robot. A robotic system bridges diverse domains such as mechatronics, computer vision, and AI. This not only requires the collaboration of experts with different backgrounds to develop a robotic system but also the experts to have a general idea of the rest of the applied disciplines. The integration of all these expertise and components developed by different groups calls for appropriate methods such as software architectures. On the other hand, eight survey respondents and one interviewee deemed the differences to other domains negligible.

Finally, we designed a list of challenges from our preliminary study in Paper A and surveyed our respondents to identify the most problematic ones—we also allowed participants to name new challenges if they deemed it necessary. In line with the perception that service robotics is a cyber-physical system domain with an emphasis on heterogeneous contexts and autonomy, the respondents agreed or strongly agreed that the most pressing challenges are achieving robustness in their robotic applications (83%), validating them (74%), and performing dynamic adaptation (62%). The solutions that are most frequently applied to such challenges are rigorous test processes involving real-world testing. As mentioned above, the lack of documentation is another key challenge for robot development in general (63%), often forcing developers to look into undocumented source code. We hypothesize that the lack of documentation in software resources might impact negatively their reuse, hindering the development pace of service robotics software engineering. Among the most common solutions provided by survey respondents to the lack of documentation, we found creating reviews, using

wikis, adding guidelines on comments, and the automatic generation of documentation.

RQ1.1. Summary of main findings

- ☞ Even though testing practices are among the most widespread quality assurance techniques in robotics, and thorough real-world testing and experimentation processes are the most performed activities, our results point out that robustness and validation are still perceived as the top challenges of the domain.
- ☞ Practitioners rely on experimentation as much and sometimes even more than on simulation.
- ☞ The development of robotic software mostly consists of integrating reusable components that implement recurrent functionalities.
- ☞ The lack of documentation in software resources is one of the most commonly mentioned challenges that might impact negatively their reuse, hindering the development pace of service robotics software engineering.
- ☞ In practice, software integration is complicated by the wide range of disciplines practitioners working to develop a robotic system have.

RQ1.2. What are the most recurrent drivers of variability in the development of service robotic applications, what practices do practitioners apply to overcome them, and what challenges do they pose?

In Paper B, we triangulate data from our experience, a multiple-case study, and a Systematic Literature Review (SLR). Thus, we were able to analyze the state-of-the-art and -practice of software variability in service robotics. We investigated three questions in detail: (i) What are the drivers of variability in the service robotics domain? (ii) What variability management practices are applied by service robotics companies to address the different drivers of variability? And (iii) what challenges do service robotics companies face when managing variability?

Through the study of Paper B we learned that *hardware* diversity, dynamic *environments*, and customized *missions* require variability in robotic applications. We identified such drivers of variability from our subject companies and described their characteristics that impact the company's practices. We learned that the environment in which a robot operates dictates the hardware the robot may equip, the standards to which it must adhere (e.g., in case the robot operates in human-populated environments), and the conditions under which it must operate robustly. The scope of robotics companies heavily determines the impact of hardware variability in their products. Companies may allow hardware customization of their products, which requires managing the variability of the robot software control systems that support such customization. On a similar note, companies targeting non-technical operators as customers may not allow them to specify missions but rather just to configure some aspects of them. The reason is their presumed lack of technical and safety regulation knowledge. Companies targeting technical operators may allow them to specify and manipulate the robots' missions, at their own risk. The company should then provide dedicated tools and mechanisms to support its customers in this task. We also learned that interaction and communication between the operator and the robots are key for the operating of the latter. To support such interaction, service robotics companies implement intuitive interfaces.

In Paper B we studied what practices are applied by service robotics companies and the literature to manage the variability that stems from the studied drivers. One of the most repeated practices through the interviews was the *installation process*, which requires the involvement of a technical operator to "install" a robotic application to the desired environment (e.g., the facilities of a customer). The installation process comprehends tasks such as mapping the environment, specifying the possible missions to be performed by the robot, and defining adaptation rules to features or events from the environment. In that regard, behavior trees [29, 52] and finite-state machines [111, 36] are mechanisms used by our studied companies to specify both missions and adaptation rules for their robotic applications. The most mentioned mechanism to manage variability by our interviewees and the SLR is configuration files. These files are used to configure robotic sensors, actuators, and software components for the robotic applications to operate in specific environments or perform specific missions. Companies parameterize the configuration files to make their software customizable and reusable and strive to make those configurations as generic as possible. Their goal is to find the "sweet spot" in which robots operate successfully without handling every single event that may occur in the environment or requiring operators to fine-tune the robots every time they are to perform a mission.

Generating generic, reusable, and parameterized configuration files that allow the robots to operate robustly requires raising the level of abstraction to make them generalizable enough, which results in a complicated task that involves many iterations of testing. The complexity of developing generic solutions is exacerbated by assumptions made by companies in terms of hardware and capabilities a robot provides—e.g., an omnidirectional navigation algorithm might not be usable by a robot with differential locomotion. Other recurrent challenges in the context of software variability in service robotics mentioned by both the companies and the literature are (i) ensuring the reliability of variant-rich systems, (ii) integrating a wide variety of hardware, electronic, and software components along with mechanics and protocols into a complex cyber-physical system as a robot, and (iii) developing user-friendly tools that manage the underlying complexity of variability while hiding it to the user.

RQ1.2. Summary of main findings

- ☞ Hardware diversity, dynamic environments, and customized missions are drivers of variability in the development of service robotic applications.
- ☞ Each driver of variability is described by characteristics such as whether hardware is customizable by customers, whether the robots' intended operating environment is populated by humans, or the mission specification mechanism provided for specific customers.
- ☞ Service robotics companies often use the same mechanisms (i.e., behavior trees, finite-state machines) to both specify robotic missions and to model the way robots must react to events from the environment.
- ☞ Configuration files are largely used by service robotics companies to manage and describe variability in their systems.
- ☞ Generating generic, reusable, and parameterized configuration files that allow robots to operate robustly with minimum manual configuration from the operator is seen as a challenge by service robotics companies.

RQ2. How can software architectures help to build decentralized and self-adaptive applications for multi-robot heterogeneous teams?

The discussion of the answer to RQ2 revolves around the problems firstly introduced in Section 1.3.2 and further explained in Paper D. These problems were inferred from the collection of empirical data we conducted with roboticists and the analysis of the literature on the topic. Each of the three problems (**P1**–**P3**) groups together a number of challenges (*Ch1*–*Ch11*) from a list proposed by Ahmad and Babar [3] and extended by us.

P1: Robotic framework. The H2020 Multi-Annual Robotics Roadmap [121] states that there is a lack of overall architectural models and methods in the robotics software development processes. This causes overhead in terms of effort a developer must invest when building robotic systems, which may be mitigated by following established, well-engineered processes. This is, in fact, in line with our findings from Papers A and B, as mentioned in the discussion of RQ1. In this context, SERA provides a number of features to tackle the first of the problems and to aid developers.

Ch 1 - Distributed Resources Access. When using SERA, each robot is considered a service with limited knowledge of the configuration and status of the rest of the team. This makes systems built with SERA scalable and enables an easy substitution of robots or the addition of new ones. This feature is introduced in Paper D and commented on in Paper G. The framework introduced in Paper H uses distributed formulations of *task models* with which each robot only models a local subset of the complete system. In order to plan a team policy in the task model, the robots need to communicate and exchange relevant information, for which they utilize ROS's communication framework.

Ch 2 - Re-engineering. SERA defines strictly fixed but well-defined interfaces among components. On one hand, this approach reduces the architecture's flexibility since the components and their interfaces cannot be easily changed. On the other hand, it greatly reduces the development time and component integration effort since developers are not concerned with communication code but just the logic of the components. This has been proved in several "Integration Weeks," in which the developing partners of the Co4Robots's consortium met to deploy and integrate their individually-developed software components into the same robotic platform with the aim of experimentation. This policy was kept thorough all the stages of SERA, as explained in Papers D, G, and H.

Ch 3 - Modeling. The first version of SERA, as described in Paper D, was conceived to automatically generate the communication code whenever a user instantiated a robotic application based on the components of the architecture. However, it has not been directly addressed yet. The framework that realizes SERA in Paper H exploits the concept of *skill models* to encode robotic capabilities in one consistent formalism to support task formulation and allocation.

Ch 4 - Design. Through all its stages, SERA has followed a component-based software engineering (CBSE) approach. The architecture encapsulates functionalities of the robotic system into reusable, modular, and exchangeable components (Papers D and G) or skills (Paper H).

Ch 5 - Programming. Each component instantiation in SERA extends an abstract class that contains methods and the communication code of each component. This, together with the application of the CBSE paradigm, enables the software platform that integrates SERA to collect a "library" of software component instantiations. The

combination of CBSE and practices from object-oriented programming ensures the correct implementation of the communication between components code and supports developers to instantiate their robotic applications or extend the platform introduced in Paper D and G. The framework of Paper H did not exploit this feature since the components that compose it—the same ones developed for SERA's platform in Papers D and G—were not expected to have different instantiations, i.e., they were fixed. Instead, the framework provides models of the robots and their capabilities using a consistent formalism. This allows the extension of the framework by implementing new robot and capability models for the task formulation and allocation.

P2: Collaboration of heterogeneous teams. As stated in the MAR [121], groups of heterogeneous robots will be required to operate in near-future industrial and social environments to carry out complex tasks. What is required for software architectures to successfully support the building of heterogeneous groups of robots?

Ch 6 - Information Fusion. In order for robots to collaborate, they need to collect, process, and share information. SERA through all its stages—Papers D, G, and H—provides specific components (i.e., *Communication and collaboration manager*, *Information Manager*, and *Localization and mapping*) with this aim. Furthermore, Paper H's framework permits that whenever new online observations are obtained, the tasks to be achieved can be re-allocated among robots. As SERA has been implemented using ROS, we made use of the MoveIt framework[25] for, among other tasks, fusing data from the robots' perception sensors.

Ch 7 - Multi-robot. To achieve complex tasks, robots may need to collaborate. For this reason, contemporary software architectures for service robots are required to provide mechanisms to support building multi-robot applications. The first two stages of SERA (as described in Papers D and G) rely on a communication approach among robots inspired by service-oriented architectures [3] using REST services. This may be easily exploited by mission specification (as demonstrated with the implementation of PROMISE in Paper E) or fleet-management tools. Paper H's framework bases the multi-robot management on a distributed task model, which makes use of the models of robotic capabilities and task specifications. As mentioned above, the framework is implemented using ROS, and therefore it exploits ROS's communication infrastructure and MoveIt for inter-robot communication and data sharing.

Ch 8 - Decentralized. The perks of decentralization for robotic applications have been listed in Section 1.4.3. SERA as described in Papers D and G works in a decentralized manner during mission execution while mission specification and distribution are performed from a centralized unit. In Paper H, the framework relies on a distributed task model to achieve decentralization.

Ch 9 - Human-robot collaboration. Robots are becoming a reality and even a commodity in certain domains. Therefore, and based on data from the MAR [121], human operators will be increasingly required to collaborate with robots in their daily jobs. This calls for reliable software architectures to build robotic applications that support such collaboration. Our architecture supports human-robot collaboration by integrating components dedicated to, for instance, recognizing gestures from an operator. Gestures may be used to alter the robotic mission or guide the robot during mission execution. This feature was tested by several means for SERA, as explained in Papers D, G, H.

P3: Self-adaptation. Service robots are often required to operate in highly dynamic environments [121], and as discussed for RQ1, it is essential for robots to implement

[25] `https://moveit.ros.org`

self-adaptive capabilities [27].

Ch 10 - Reconfiguration. A common strategy in robotics to allow robots to robustly operate in dynamic environments is to instill them with the capability of reconfiguring their components and behavior. SERA's first stages, as described in Papers D and G, support the components reconfiguration by following a MAPE-K-like approach [30]. For instance, during one of our Integration Weeks, we detected the high-computation requirements of the gesture-recognition component. This affected the overall performance of the robot, sometimes causing errors in the collision-avoidance algorithm, which led to physical crashes. Therefore, we provided SERA with a mechanism to load and unload instantiations of its components as plugins. However, the first stages of SERA lacked the runtime monitoring activity within the MAPE-K loop [64] and therefore the loading and unloading of components' instantiations were performed manually. On the other hand, the reconfiguration of robotic behavior is mainly documented in Paper E. Three components of SERA support the autonomous behavior adaptation at different levels through reconfiguration, namely: (i) the *Local mission manager* performs local mission adaptation (e.g., the robot should go to its charging station if it detects a low battery level); (ii) the *Adaptation manager* perform tasks' adaptation, as high-level replanning (e.g., there is a closed door in the robot's path so the path needs to be replanned); (iii) the *Motion planning and control* performs low-level replanning (e.g., collision avoidance). In Paper H, the adaptation is achieved through an auction approach to re-allocate tasks among the robots that are part of the team based on their observations.

Ch 11 - Fault Tolerance. Paper H's framework allows the re-allocation of robots to other tasks based on their capabilities in case of fault identification.

RQ2. Summary of main findings

- ☞ SERA as presented in Paper H addresses all the challenges identified by Ahmad and Babar [3] and those by us.
- ☞ SERA exploits the CBSE paradigm to promote modularity, reusability, and exchangeability of its components.
- ☞ SERA supports robot-robot and human-robot collaboration by providing appropriate components that support sharing information and updates in their mission through gestures from a human operator at run time.
- ☞ Mission execution is performed in a distributed manner by robotic applications built with SERA.
- ☞ SERA provides self-adaptive capabilities to robotic applications through the usage of appropriate software components and models.

RQ3. How can tools and mechanisms such as Domain-Specific Languages and Integrated Development Environments support operators specifying the intended behaviors of robotic systems?

To answer RQ3 we conducted a combination of a knowledge-seeking study (Paper C) and three solution-seeking studies (Papers E, G, and H), which in turn helped us to answer RQ3.1 and RQ3.2, respectively.

As stated by the MAR [121] and confirmed by Paper B, in order to reduce costs and establish a vibrant component market in service robotics, there is a need for instruments that support mission *reuse* and diversification that at the same time allow coping with

the *variability* of conditions of application scenarios. Furthermore, we identified through the study of Paper A that the complexity of mission specification often does not reside in commanding a robot with a set of tasks but in making the robotic application robust enough to cope with the variability that characterizes the real environments in which the robots are required to operate, especially those that involve humans.

Another concern we identified through the reviews of the literature we conducted in both Papers C and E, is that most articles in Domain-Specific Languages (DSLs) for robotics focus on the technical aspects and experimentation of the language and rarely on its implementation process. The community would require engineers and researchers to conceive new DSLs that capture and simplify the specification of the concerns of operators as well as papers that detail the DSL's development process [118]. This should include the description of involved stakeholders and their role in the development, the activities in which they take part (e.g., extraction and analysis of subdomains, description of use cases, tooling implementation), and the artifacts and models that are generated from those activities.

RQ3.1. What Domain-Specific Languages and Integrated Development Environments exist to support the specification of mission and behaviors of robotic systems in end-user domains?

Through the study of the Integrated Development Environments (IDEs) and DSLs for robotic mission specification in Paper C, we discovered that the community has made substantial progress in the field. In the last years, mission specification languages and end-user-oriented mission environments have been proposed both from the research community and industry. However, we concluded that the mission specification problem still requires solutions able to make robots usable in everyday life for accomplishing complex missions. Concretely, we identified a list of themes that in our opinion would require further investigation from researchers and practitioners to tackle the challenges stated throughout the discussion of RQ3.

- *Reusability*. DSLs could make use of reusable libraries of tasks and skills, including libraries from other projects and initiatives, such as RobMoSys [1].

- *Explicit specification of concerns*. DSLs could allow (potentially non-technical) operators to specify explicitly: (i) the variability of conditions of real-world scenarios, and (ii) requirements and constraints—e.g., safety standards[26]—given by, for example, the robots' operating environment.

- *Autonomous mission allocation*. Instead of requesting (potentially non-technical) operators to specify robot-specific local missions, a mechanism integrated into the DSL could automatically assign and potentially re-assign robots to missions during mission execution according to their capabilities and various quality parameters. In other words, operators would only be concerned with specifying the overall mission and the DSL would decompose it autonomously.

- *Human-robot collaboration*. DSLs could support specifying missions that require collaboration with humans.

[26] `https://www.york.ac.uk/assuring-autonomy/projects/unmanned-aerial-vehicles-safety/`

RQ3.1. Summary of main findings

☞ Many solutions from both academia and companies have been proposed in the last years for mission specification in robotics. Most solutions consist of DSLs that incorporate descriptions of missions in natural language and visual and end-user-oriented mission environments.

☞ We hypothesize that to support mission *reuse* and diversification while at the same time allow coping with the *variability* of conditions of application scenarios occurring in real environments, more investigation needs to be conducted in four themes, namely (i) reusability, (ii) explicit specification of concerns, (iii) autonomous mission allocation, and (iv) human-robot collaboration.

RQ3.2. How to support operators in the specification of complex missions for teams of heterogeneous robots combining expressiveness, simplicity, and rigorousness?

In Paper E, we analyzed several methods that have been used in the robotics community for mission specification. We evaluated all these mission specification methods, based on three features:

- *Expressiveness*. Mission specification must be expressive enough to model a variety of behaviors and actions. An expressive language allows for a richer definition of concerns by the user, for instance recovering after errors.

- *Simplicity*. In the scenarios envisioned by the H2020 Multi-Annual Robotics Roadmap (MAR) [121], mission specification methods must promote simplicity to support users without deep knowledge of programming languages—i.e., non-technical operators. This was corroborated by our knowledge-seeking study of Paper B, as the development of user-friendly mission specification mechanisms was considered one of the most pressing challenges by our interviewees.

- *Rigorousness*. Rigorous languages ensure that mission specification precisely and unambiguously represents the mission to be executed.

In our understanding, mission-specification languages should promote these qualities.

The patterns and operators on which this work is based promote the expressiveness of PROMISE since they enable specifying complex missions with their combination. The expressiveness of PROMISE was validated by experimentation, defining a series of complex missions formulated for a well-known robotics competition (RoboCup@Home [93]), and a home-grown one in Paper E. The expressiveness and rigorousness of PROMISE along with its implementation was evaluated through experimentation in several real-world scenarios with real robots, as documented in Papers G and H. As the target user for PROMISE are domain experts in robotics—i.e., operators with expertise in robotics but who may not have advanced knowledge about programming languages—, simplicity is promoted by tailoring the terminology of our language to that of the domain of robotics and by conceiving operators that do not require knowledge of programming languages for their usage. Also, our goal was to develop a declarative language so the user could focus on specifying the goal of the mission instead of the required steps to satisfy it. We evaluated the simplicity of

PROMISE by conducting two user studies from which we collected the perception of the participants on the user-friendliness of the language and its implementation. We strove to enhance the trade-off between expressiveness and simplicity of our language (i.e., usability) while keeping rigorous semantics. The rigorousness of PROMISE is granted by the use of LTL as the underlying language used by the planning component. Nevertheless, users do not need to concern with the semantics and syntax of LTL, inasmuch as PROMISE raises the level of abstraction from such temporal logic.

As stated above, to accomplish complex tasks, robots with different capabilities might be required to operate collaboratively. Hence, mission specification methods will be required to support teams of heterogeneous robots. To this end, PROMISE and its toolchain were designed to be platform-agnostic, only requiring making a few changes in the toolchain's interpreter when adding a new robot platform to the application.

RQ3.2. Summary of main findings

- ☞ PROMISE was evaluated to provide a combination of expressiveness, simplicity, and rigorousness as a mission specification language.
- ☞ After a training of 75 minutes divided in two sessions, all of the participants to the second user study were able to correctly specify all the missions we proposed.
- ☞ Using PROMISE, a trained user can successfully specify all the missions from a well-known robotics competition such as RoboCup@Home.
- ☞ The users from our user studies perceived the language of PROMISE more satisfactory than its implementation.
- ☞ We believe that a better implementation of PROMISE and the integration of subdomain-specific operators would further improve the perception from users in terms of expressiveness and simplicity.

RQ4. What are the applicable strategies to manage heterogeneous robotic applications with only partial knowledge of dynamic environments?

To evaluate the underlying theory of MAPmAKER, we developed a proof-of-concept tool using MATLAB that is built upon the planner proposed by Tumova et al. [127]. We conducted two evaluations aiming to assess (i) the performance of MAPmAKER in scenarios with only partial information and (ii) the usability of MAPmAKER to heterogeneous groups of robots. To this end, we evaluated the ability of MAPmAKER to compute plans in partially known environments and to what extent the employed decentralized algorithm reduces computation overhead. We evaluated MAPmAKER through experimentation, the setting was as follows. We conducted three experiments; one for each aspect of robotic missions that could be affected by uncertainty, as described in Section 1.4.5. The experiments took place in two simulated real-world environments. The first of them was found in the RoboCup Logistics League [75] and the second represents an apartment where a number of robotics projects have been applied to assist elderly people [4]. We considered for the evaluation of MAPmAKER nine different scenarios, which result from a combination of three configurations of the robot models in terms of partial information and three initial conditions for the robots within the robotic application. For all scenarios, we included two robots in the robotic application, since this is the number typically used in the scenarios within the RoboCup Logistics League.

For the first evaluation (i.e., assessing the performance of MAPmAKER in scenarios with partial information), we performed two steps, both of them considering the nine scenarios explained above. In the first step, we run MAPmAKER allowing the computation of possible plans in all scenarios. In the second step, unknown transitions, services, and meeting capabilities were set as not executable, not provided, and not possible, respectively. We performed measurements in the time computation and length of the plans of each scenario in both steps so as to compare the results. Based on the results, we concluded that MAPmAKER is effective whenever it computes a possible plan and during its execution a *true* evidence of partial information is detected—i.e., a robot discovers that a piece of partial information of the model is true. When no partial information was involved in the plans computed by MAPmAKER, they had the same length as the plans generated by a classical planner (as the one presented by Tumova et al. [127]). In several scenarios, MAPmAKER allowed the achievement of the mission while a classical procedure could not. MAPmAKER was able to compute possible plans when no definitive plan was available and true evidence about partial information was detected during the plan execution. Finally, the detection of a *false* evidence—i.e., a robot discovers that a piece of partial information of the model is false—decreases the effectiveness of MAPmAKER. This is caused by the need for recomputing the plans for the robots.

For the second evaluation (i.e., assessing whether MAPmAKER improves computation of plans for heterogeneous robotic applications), we run the same nine scenarios (consisting of the same combination of robot partial models and initial conditions) in the same two simulated environments. To assess whether the decentralized procedure helps with the planning computation, we added a third robot to the robotic application. The third robot was not required to meet with the two other robots and therefore the decentralized procedure split the robotic application into two subclasses—one with the first two robots that must meet to achieve their missions and a second one with the third and independent robot. The second evaluation also consisted of two steps: in the first one we enabled the decentralized procedure that divides the robotic team into subclasses, and in the second we disabled it. We set a time frame of one hour for the planner to compute and execute the plans. For all the scenarios of the first step, MAPmAKER was able to finish its execution within the given time frame of one hour. As opposed, MAPmAKER could not finish the computing and execution of plans within the time frame for the second evaluation. We conclude that decentralized procedures are of paramount importance to implement planning solutions in real-world robotic scenarios that require scalability.

RQ4. Summary of main findings

- ☞ MAPmAKER allows robotic applications to achieve their missions in scenarios with partial information, while a classical procedure could not.
- ☞ In scenarios without partial information, plans computed by MAPmAKER have the same length as plans generated by a classical planner.
- ☞ MAPmAKER is effective whenever it computes a possible plan and during its execution a *true* evidence of partial information is detected.
- ☞ Detecting a *false* evidence during mission execution decreases MAPmAKER's effectiveness, since it requires recomputing the plans.
- ☞ MAPmAKER's decentralized procedure permits a faster computation of plans for heterogeneous robotic applications than a centralized approach.

1.6 Conclusion and Future Work

This thesis aims at helping consolidate the community of service robotics software engineering to substantially lift the state-of-practice for engineering service robotic applications. The goal is to investigate practices and challenges in service robotics software engineering to provide guidance for practitioners and researchers as well as to engineer solutions for some of the challenges we identified. To fulfill this goal, we conducted a combination of three knowledge-seeking studies and five solution-seeking studies following a methodology inspired by the design-science paradigm. Our knowledge-seeking studies identify limitations, challenges, and problems in the domain of service robotics software engineering. In the five solution-seeking studies we engineered solutions for some of the identified problems. In particular, we study:

- Architectural models to enable and enhance aspects of systematic development of robotic applications (e.g., modularity, reusability).
- Robotic mission specification methods, focusing on domain-specific languages.
- Decentralized planning solutions that allow robotic applications to plan and operate in environments with the presence of uncertainty.

Besides these problems, we envision three lines of future work:

Feature-modeling architecture. In the study of Papers B and D, we learned about the complexity of managing variability for service robotics companies. Service robotics is a fast-paced domain in terms of growth, and according to Moore's Laws [16] it will only become more complex with time. Thus, robotics companies will increasingly need to manage variability when it comes to complex and computationally expensive tasks. For these reasons, we will extend SERA (as proposed in Papers D and G) to support variability in a domain where variability is typically performed in an ad hoc manner. The main goal of this study is to provide a framework based on the software product line engineering paradigm to support roboticists with a guided means of customizing their products and model bindings of their features. In summary, the proposed framework implementation would alleviate extension complexity, reduce performance costs, and minimize resource consumption in robotic systems instantiation and customization.

Improvements in PROMISE. To tackle some of the identified limitations from our Domain-Specific Language (DSL), we will: 1) refine the current implementation of the DSL to solve some of the problems stated by the participants of the second conducted user study; 2) create support for run-time changes to a mission specification; 3) conduct further user studies with participants with different expertise to assess the simplicity of PROMISE and to collect users' feedback; and 4) investigate the optimal ways for supporting the synchronization among robots.

Mission specification mechanisms in other domains. Some of the studies conducted for this thesis demonstrate and discuss the importance of DSLs for mission specification in robotics. We propose applying the knowledge acquired from this thesis to use those languages in other domains. Concretely, we see a huge potential in applying behavior trees to autonomous vehicles. Among other benefits, this would allow engineers to specify realistic behaviors for autonomous vehicles and other agents in test-case scenarios.

Bibliography

[1] "The RobMoSys project," https://robmosys.eu, 2020.

[2] P. E. Agre and D. Chapman, "What are plans for?" *Robotics and autonomous systems*, vol. 6, no. 1-2, pp. 17–34, 1990.

[3] A. Ahmad and M. A. Babar, "Software architectures for robotic systems: A systematic mapping study," *Journal of Systems and Software*, vol. 122, pp. 16–39, 2016.

[4] Ängen Research & Innovation, "The Ängen research and innovation apartment," http://angeninnovation.se, 2014.

[5] S. Apel, D. Batory, C. Kästner, and G. Saake, *Feature-Oriented Software Product Lines*. Springer, Berlin Heidelberg, 2013.

[6] M. Babaei, K. Jahed, and J. Dingel, "Solving the robosoccer challenge problem with uml-rt and papyrus-rt," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2019, pp. 221–227.

[7] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[8] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.

[9] M. Ben-Ari, A. Pnueli, and Z. Manna, "The temporal logic of branching time," *Acta informatica*, 1983.

[10] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wasowski, "A survey of variability modeling in industrial practice," in *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, 2013, pp. 1–8.

[11] T. Berger, R.-H. Pfeiffer, R. Tartler, S. Dienst, K. Czarnecki, A. Wasowski, and S. She, "Variability mechanisms in software ecosystems," *Information and Software Technology*, vol. 56, no. 11, pp. 1520–1535, 2014.

[12] T. Berger, D. Lettner, J. Rubin, P. Grünbacher, A. Silva, M. Becker, M. Chechik, and K. Czarnecki, "What is a feature? a qualitative study of features in industrial software product lines," in *Proceedings of the 19th International Conference on Software Product Line*, 2015, pp. 16–25.

[13] T. Berger, J.-P. Steghöfer, T. Ziadi, J. Robin, and J. Martinez, "The state of adoption and the challenges of systematic variability management in industry," *Empirical Software Engineering*, vol. 25, pp. 1755–1797, 2020.

[14] A. Bhave, B. H. Krogh, D. Garlan, and B. Schmerl, "View consistency in architectures for cyber-physical systems," in *2011 IEEE/ACM second international conference on cyber-physical systems*. IEEE, 2011, pp. 151–160.

[15] J. Bohren and S. Cousins, "The smach high-level executive [ros news]," *IEEE Robotics & Automation Magazine*, vol. 17, no. 4, pp. 18–20, 2010.

[16] R. Bourgonjon, "Embedded systems in consumer products," in *School organized by the European Educational Forum*. Springer, 1996, pp. 395–403.

[17] D. Bozhinoski, D. Di Ruscio, I. Malavolta, P. Pelliccione, and I. Crnkovic, "Safety for mobile robotic systems: A systematic mapping study from a software engineering perspective," *Journal of Systems and Software*, vol. 151, pp. 150–179, 2019.

[18] V. Braberman, N. D'Ippolito, J. Kramer, D. Sykes, and 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.

[19] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis lectures on software engineering*, vol. 3, no. 1, pp. 1–207, 2017.

[20] R. Brooks, "A robust layered control system for a mobile robot," *IEEE journal on robotics and automation*, vol. 2, no. 1, pp. 14–23, 1986.

[21] D. Brugali and E. Prassler, "Software engineering for robotics," *Robotics & Automation Magazine, IEEE*, vol. 16, no. 1, pp. 9–15, 2009.

[22] D. Brugali, "Model-driven software engineering in robotics: Models are designed to use the relevant things, thereby reducing the complexity and cost in the field of robotics," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 155–166, 2015.

[23] D. Brugali and M. Reggiani, "Software stability in the robotics domain: issues and challenges," in *International Conference on Information Reuse and Integration*. IEEE, 2005.

[24] D. Brugali, A. Brooks, A. Cowley, C. Côté, A. C. Domínguez-Brito, D. Létourneau, F. Michaud, and C. Schlegel, "Trends in component-based robotics," in *Software Engineering for Experimental Robotics*. Springer, 2007, pp. 135–142.

[25] D. Brugali, R. Capilla, and M. Hinchey, "Dynamic variability meets robotics," *Computer*, vol. 48, no. 12, pp. 94–97, 2015.

[26] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali, "The brics component model: a model-based development paradigm for complex robotics software systems," in *Proceedings*

of the 28th Annual ACM Symposium on Applied Computing. ACM, 2013, pp. 1758–1764.

[27] J. Cámara, B. Schmerl, and 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.

[28] J. S. Cepeda, R. Soto, J. L. Gordillo, and L. Chaimowicz, "Towards a service-oriented architecture for teams of heterogeneous autonomous robots," in *2011 10th Mexican International Conference on Artificial Intelligence*. IEEE, 2011, pp. 102–108.

[29] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[30] A. Computing *et al.*, "An architectural blueprint for autonomic computing," *IBM White Paper*, vol. 31, no. 2006, pp. 1–6, 2006.

[31] E. Coste-Maniere and R. Simmons, "Architecture, the backbone of robotic systems," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1. IEEE, 2000, pp. 67–72.

[32] J. W. Creswell and J. D. Creswell, *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2017.

[33] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Boston, MA: Addison-Wesley, 2000.

[34] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wasowski, "Cool features and tough decisions: a comparison of variability modeling approaches," in *Proceedings of the sixth international workshop on variability modeling of software-intensive systems*, 2012, pp. 173–182.

[35] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel *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.

[36] S. Dragule, T. Berger, C. Menghi, and P. Pelliccione, "A survey on the design space of end-user oriented languages for specifying robotic missions," *International Journal of Software and Systems Modeling (SoSYM).*, 2021.

[37] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An exploratory study of cloning in industrial software product lines," in *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 2013, pp. 25–34.

[38] T. Dunn and R. Erlich, *The mechanical god: Machines in science fiction*. Greenwood Press, 1982.

[39] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to advanced empirical software engineering*. Springer, 2008, pp. 285–311.

[40] G. Edwards, J. Garcia, H. Tajalli, D. Popescu, N. Medvidovic, G. Sukhatme, and B. Petrus, "Architecture-driven self-adaptation and self-management in robotics systems," in *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 2009, pp. 142–151.

[41] E. Engström, M.-A. Storey, P. Runeson, M. Höst, and M. T. Baldassarre, "How software engineering research aligns with design science: a review," *Empirical Software Engineering*, vol. 25, no. 4, pp. 2630–2660, 2020.

[42] P. Estefo, J. Simmonds, R. Robbes, and J. Fabry, "The robot operating system: package reuse and community dynamics," *Journal of Systems and Software*, vol. 151, pp. 226–242, 2019.

[43] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[44] C. Finucane, G. Jing, and H. Kress-Gazit, "LTLMoP: Experimenting with language, temporal logic and robot control," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2010, pp. 1988–1993.

[45] R. J. Firby, "An investigation into reactive planning in complex domains." in *AAAI*, vol. 87, 1987, pp. 202–206.

[46] M. Fowler, *Domain-specific languages*. Pearson Education, 2010.

[47] M. Galster, P. Avgeriou, D. Weyns, and T. Männistö, "Variability in software architecture: current practice and challenges," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 5, pp. 30–32, 2011.

[48] S. García, P. Pelliccione, C. Menghi, T. Berger, and T. Bures, "High-level mission specification for multiple robots," in *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering*, 2019, pp. 127–140.

[49] S. García, D. Strüber, D. Brugali, T. Berger, and P. Pelliccione, "Robotics software engineering: A perspective from the service robotics domain," in *Proceedings of the 2020 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020.

[50] D. Garlan and M. Shaw, "An introduction to software architecture," in *Advances in software engineering and knowledge engineering*. World Scientific, 1993, pp. 1–39.

[51] J. C. Georgas and R. N. Taylor, "Policy-based self-adaptive architectures: a feasibility study in the robotics domain," in *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, 2008, pp. 105–112.

[52] R. Ghzouli, T. Berger, E. B. Johnsen, S. Dragule, and A. Wasowski, "Behavior trees in action: A study of robotics applications," in *13th ACM SIGPLAN International Conference on Software Language Engineering (SLE)*, 2020.

[53] F. Giunchiglia and P. Traverso, "Planning as model checking," in *European Conference on Planning*. Springer, 1999, pp. 1–20.

[54] M. Gombolay, X. J. Yang, B. Hayes, N. Seo, Z. Liu, S. Wadhwania, T. Yu, N. Shah, T. Golen, and J. Shah, "Robotic assistance in the coordination of patient care," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1300–1316, 2018.

[55] S. Götz, M. Leuthäuser, J. Reimann, J. Schroeter, C. Wende, C. Wilke, and U. Aßmann, "A role-based language for collaborative robot applications," in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2011, pp. 1–15.

[56] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local LTL specifications," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015.

[57] M. Guo, K. H. Johansson, and D. Dimarogonas, "Revising motion planning under linear temporal logic specifications in partially known workspaces," in *International Conference on Robotics and Automation*. IEEE, 2013.

[58] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal aspects of computing*, vol. 6, no. 5, pp. 512–535, 1994.

[59] D. Harel and M. Politi, *Modeling reactive systems with statecharts: the STATEMATE approach*. McGraw-Hill, Inc., 1998.

[60] B. Hayes and B. Scassellati, "Effective robot teammate behaviors for supporting sequential manipulation tasks," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 6374–6380.

[61] F. W. Heckel, G. M. Youngblood, and N. S. Ketkar, "Representational complexity of reactive agents," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. IEEE, 2010, pp. 257–264.

[62] A. K. Hemal and M. Menon, *Robotics in genitourinary surgery*. Springer, 2018.

[63] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, pp. 75–105, 2004.

[64] N. Hili, M. Bagherzadeh, K. Jahed, and J. Dingel, "A model-based architecture for interactive run-time monitoring," *Software and Systems Modeling*, pp. 1–23, 2020.

[65] IFR, "IFR - Robot history," https://ifr.org/robot-history, 2019.

[66] ——, "IFR - Industrial robots," https://ifr.org/downloads/press2018/Executive_Summary_WR_2018_Industrial_Robots.pdf, 2018.

[67] ——, "IFR - Service robots," https://ifr.org/downloads/press2018/Executive_Summary_WR_Service_Robots_2018.pdf, 2018.

[68] E.-M. Ihantola and L.-A. Kihn, "Threats to validity and reliability in mixed methods accounting research," *Qualitative Research in Accounting & Management*, 2011.

[69] J. F. Inglés-Romero, C. Vicente-Chicote, B. Morin, and O. Barais, "Towards the automatic generation of self-adaptive robotics software: An experience report," in *2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. IEEE, 2011, pp. 79–86.

[70] D. Isla, "Handling Complexity in the Halo 2 AI," in *In Game Developers Conference*, 2005.

[71] ISO, "ISO - Robotics," https://www.iso.org/obp/ui/\#iso:std:iso:8373:ed-2:v1:en, 2012.

[72] ISO, "ISO/PAS 21448:2019. Road vehicles – Safety of the intended functionality," https://www.iso.org/standard/70939.html, 2019.

[73] ——, "ISO 8373:2012. Robots and robotic devices – Vocabulary," https://www.iso.org/standard/55890.html, 2012.

[74] M. Y. Jung, A. Deguet, and P. Kazanzides, "A component-based architecture for flexible integration of robotic systems," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 6107–6112.

[75] C. D. U. Karras, T. Neumann, T. N. A. Rohr, W. Uemura, D. Ewert, N. Harder, S. Jentzsch, N. Meier, and S. Reuter, "Robocup logistics league rules and regulations," 2016.

[76] S. Kernbach, D. Häbe, O. Kernbach, R. Thenius, G. Radspieler, T. Kimura, and T. Schmickl, "Adaptive collective decision-making in limited robot swarms without communication," *The International Journal of Robotics Research*, vol. 32, no. 1, pp. 35–55, 2013.

[77] J. Kitzinger, "Qualitative research: introducing focus groups," *BMJ*, vol. 311, no. 7000, pp. 299–302, 1995.

[78] M. Klotzbücher and H. Bruyninckx, "Coordinating robotic tasks and systems with rfsm statecharts," 2012.

[79] M. Klotzbücher, R. Smits, H. Bruyninckx, and J. De Schutter, "Reusable hybrid force-velocity controlled motion specifications with executable domain specific languages," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 4684–4689.

[80] D. E. Koditschek and E. Rimon, "Robot navigation functions on manifolds with boundary," *Advances in applied mathematics*, vol. 11, no. 4, pp. 412–442, 1990.

[81] D. Kortenkamp, R. Simmons, and D. Brugali, "Robotic systems architectures and programming," in *Springer Handbook of Robotics*. Springer, 2016, pp. 283–306.

[82] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *Future of Software Engineering (FOSE'07)*. IEEE, 2007, pp. 259–268.

[83] H. Kress-Gazit, "Robot challenges: Toward development of verication and synthesis techniques [errata]," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 108–109, 2011.

[84] T. R. Kurfess, *Robotics and automation handbook*. CRC press, 2004.

[85] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer, 2007, pp. 220–270.

[86] ——, "Prism 4.0: Verification of probabilistic real-time systems," in *International conference on computer aided verification*. Springer, 2011, pp. 585–591.

[87] S. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[88] A. Liekna, E. Lavendelis, and A. Nikitenko, "Challenges in development of real time multi-robot system using behaviour based agents," in *Distributed Computing and Artificial Intelligence*. Springer, 2013, pp. 587–595.

[89] S. R. Lindemann, I. I. Hussein, and S. M. LaValle, "Real time feedback control for nonholonomic mobile robots with obstacles," in *Proceedings of the 45th IEEE Conference on Decision and Control*, Dec 2006, pp. 2406–2411.

[90] D. C. MacKenzie, J. M. Cameron, and R. C. Arkin, "Specification and execution of multiagent missions," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1995.

[91] W. Mahmood, D. Strüber, T. Berger, R. Laemmel, and M. Mukelabai, "Seamless variability management with the virtual platform," in *43rd International Conference on Software Engineering (ICSE)*, 2021.

[92] S. Maoz and J. O. Ringert, "GR(1) synthesis for LTL specification patterns," in *Foundations of Software Engineering (FSE)*. ACM, 2015.

[93] M. Matamoros, C. Rascon, J. Hart, D. Holz, and L. van Beek, "RoboCup@Home 2018: Rules and Regulations," http://www.robocupathome.org/rules/2018_rulebook.pdf, 2018.

[94] N. C. Mendonça, D. Garlan, B. Schmerl, and J. Cámara, "Generality vs. reusability in architecture-based self-adaptation: the case for self-adaptive microservices," in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, 2018, pp. 1–6.

[95] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger, "Specification patterns for robotic missions," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[96] C. Menghi, S. García, P. Pelliccione, and J. Tumova, "Multi-robot LTL planning under uncertainty," in *International Symposium on Formal Methods*. Springer, 2018, pp. 399–417.

[97] C. Menghi, C. Tsigkanos, T. Berger, P. Pelliccione, and C. Ghezzi, "Property specification patterns for robotic missions," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*. IEEE, 2018, pp. 434–435.

[98] C. Menghi, C. Tsigkanos, T. Berger, and P. Pelliccione, "PsALM: Specification of dependable robotic missions," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 99–102.

[99] D. J. Miller and R. C. Lennox, "An object-oriented environment for robot system architectures," *IEEE Control Systems Magazine*, vol. 11, no. 2, pp. 14–23, 1991.

[100] P. J. Mosterman and J. Zander, "Cyber-physical systems challenges: a needs analysis for collaborating embedded software systems," *Software & Systems Modeling*, vol. 15, no. 1, pp. 5–16, 2016.

[101] D. Nešić, J. Krüger, t. Stănciulescu, and T. Berger, "Principles of feature modeling," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 62–73.

[102] I. A. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I.-H. Shu *et al.*, "Claraty: Challenges and steps toward reusable robotic software," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, p. 5, 2006.

[103] A. Nordmann, N. Hochgeschwender, and S. Wrede, "A survey on domain-specific languages in robotics," in *Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2014, pp. 195–206.

[104] A. Nordmann, N. Hochgeschwender, D. L. Wigand, and S. Wrede, "A Survey on Domain-Specific Modeling and Languages in Robotics," *Journal of Software Engineering in Robotics (JOSER)*, vol. 7, no. 1, pp. 75–99, 2016.

[105] H. Oestreich, S. Wrede, and B. Wrede, "Learning and performing assembly processes: an overview of learning and adaptivity in digital assistance systems for manufacturing," in *Proceedings of the 13th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, 2020, pp. 1–8.

[106] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Software engineering notes*, vol. 17, no. 4, pp. 40–52, 1992.

[107] J. L. Peterson, "Petri nets," *ACM Computing Surveys (CSUR)*, vol. 9, no. 3, pp. 223–252, 1977.

[108] A. Pnueli, "The temporal logic of programs," in *Foundations of Computer Science*. IEEE, 1977.

[109] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[110] A. Rajhans, A. Bhave, I. Ruchkin, B. H. Krogh, D. Garlan, A. Platzer, and B. Schmerl, "Supporting heterogeneity in cyber-physical systems architectures," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3178–3193, 2014.

[111] M. Risler and O. von Stryk, "Formal behavior specification of multi-robot systems using hierarchical state machines in xabsl," in *AAMAS08-workshop on formal models and methods for multi-robot systems*. Citeseer, 2008, pp. 12–16.

[112] P. Runeson, E. Engström, and M.-A. Storey, "The design science paradigm as a frame for empirical software engineering," in *Contemporary empirical methods in software engineering*. Springer, 2020, pp. 127–147.

[113] D. D. Ruscio, I. Malavolta, P. Pelliccione, and M. Tivoli, "Automatic generation of detailed flight plans from high-level mission descriptions," in *MODELS*, 2016.

[114] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Hierarchical ltl-task mdps for multi-agent coordination through auctioning and learning," *The international journal of robotics research*, 2019.

[115] C. Schlegel, T. Haßler, A. Lotz, and A. Steck, "Robotic software systems: From code-driven to model-driven designs," in *2009 International Conference on Advanced Robotics*. IEEE, 2009, pp. 1–8.

[116] B. Schmerl and D. Garlan, "Acmestudio: Supporting style-centered architecture development," in *Proceedings of the 26th International Conference on Software Engineering*. IEEE Computer Society, 2004, pp. 704–705.

[117] D. Schmidt, "Guest editor's introduction: Model-driven engineering," *Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[118] S. Schneider, N. Hochgeschwender, and G. K. Kraetzschmar, "Structured design and development of domain-specific languages in robotics," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2014, pp. 231–242.

[119] M. Schoppers, "Universal plans for reactive robots in unpredictable environments." in *IJCAI*, vol. 87. Citeseer, 1987, pp. 1039–1046.

[120] B. Schwartz, L. Nägele, A. Angerer, and B. A. MacDonald, "Towards a graphical language for quadrotor missions," *arXiv preprint arXiv:1412.1961*, 2014.

[121] SPARC, "Robotics 2020 Multi-Annual Roadmap," https://eu-robotics.net/sparc/upload/about/files/H2020-Robotics-Multi-Annual-Roadmap-ICT-2016.pdf, 2016.

[122] A. Steck, A. Lotz, and C. Schlegel, "Model-driven engineering and run-time model-usage in service robotics," in *ACM SIGPLAN Notices*, vol. 47, no. 3, 2011, pp. 73–82.

[123] G. Steinbauer, "A survey about faults of robots used in robocup," in *Robot Soccer World Cup*. Springer, 2012, pp. 344–355.

[124] K.-J. Stol and B. Fitzgerald, "The abc of software engineering research," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 3, p. 11, 2018.

[125] K.-J. Stol, M. Goedicke, and I. Jacobson, "Introduction to the special section—general theories of software engineering: New advances and implications for research," *Information and Software Technology*, vol. 70, pp. 176–180, 2016.

[126] U. Thomas, G. Hirzinger, B. Rumpe, C. Schulze, and A. Wortmann, "A new skill based robot programming language using uml/p statecharts," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 461–466.

[127] J. Tumova and D. V. Dimarogonas, "Multi-agent planning under local LTL specifications and event-based synchronization," *Automatica*, vol. 70, pp. 239–248, 2016.

[128] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimal multi-robot path planning with temporal logic constraints," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011.

[129] J. E. Van Aken, "Management research as a design science: Articulating the research products of mode 2 knowledge production in management," *British journal of management*, vol. 16, no. 1, pp. 19–36, 2005.

[130] F. J. Van der Linden, K. Schmid, and E. Rommes, *Software product lines in action: the best industrial practice in product line engineering*. Springer Science & Business Media, 2007.

[131] A. Van Deursen, P. Klint, and J. Visser, "Domain-specific languages: An annotated bibliography," *ACM Sigplan Notices*, vol. 35, no. 6, pp. 26–36, 2000.

[132] T. Vogel, "mrubis: An exemplar for model-based architectural self-healing and self-optimization," in *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*, 2018, pp. 101–107.

[133] F.-Y. Wang, K. J. Kyriakopoulos, A. Tsolkas, and G. N. Saridis, "A petri-net coordination model for an intelligent mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 4, pp. 777–789, 1991.

[134] C. C. White, "A survey of solution techniques for the partially observed markov decision process," *Annals of Operations Research*, vol. 32, no. 1, pp. 215–230, 1991.

[135] J. Wienke and S. Wrede, "Results of the survey: failures in robotics and intelligent systems," *arXiv preprint arXiv:1708.07379*, 2017.

[136] R. J. Wieringa, *Design science methodology for information systems and software engineering*. Springer, 2014.

[137] E. M. Wolff, U. Topcu, and R. M. Murray, "Automaton-guided controller synthesis for nonlinear systems with temporal logic," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013.

[138] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, 2013.

[139] M. Yannakakis, "Hierarchical state machines," in *IFIP International Conference on Theoretical Computer Science*. Springer, 2000, pp. 315–330.

[140] V. A. Ziparo, L. Iocchi, D. Nardi, P. F. Palamara, and H. Costelha, "Petri net plans: a formal model for representation and execution of multi-robot plans," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 79–86.