GRAN SASSO SCIENCE INSTITUTE
CENTER FOR ADVANCED STUDIES INFN

Thesis Dissertation

Managing safety and run-time adaptability in mission-critical Mobile Multi-Robot Systems

Author:

Darko Bozhinoski
darko.bozhinoski@gssi.infn.it

Supervisors:

Patrizio Pelliccione
patrizio.pelliccione@gu.se

Ivano Malavolta
ivano.malavolta@gssi.infn.it

December 2017

GSSI Gran Sasso Science Institute
Viale Francesco Crispi, 7 - 67100 L'Aquila - Italy

Declaration of Authorship

I, DARKO BOZHINOSKI, declare that this thesis titled, Managing safety and run-time adaptability in mission-critical Mobile Multi-Robot Systems and the work presented in it are my own. I confirm that:

- This thesis has been composed by myself and the presented work is my own under the guidance of my supervisors Patrizio Pelliccione and Ivano Malavolta for a research degree at this University.
- Moreover, Chapter 2 is essentially [1], Chapter 3 is essentially [2] co-authored with Ruscio, Di D., Malavolta, I., Pelliccione P., & Crnkovic I.; Chapter 4 is re-elaboration of [3] co-authored with D., Di Ruscio, D., Malavolta, I., Pelliccione, P., & Tivoli, M.; Chapter 5 is based on [4] co-authored with Bucchiarone, A., Malavolta, I., Marconi, A., & Pelliccione, P; and Chapter 6 is based on [5] co-authored with Garlan, D., Malavolta, I., & Pelliccione, P.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself

Abstract

The coming future will be pervaded by mobile multi-robot systems (MMRSs) that will facilitate tasks of everyday life and open new business and societal opportunities. However, these systems are exposed to various spheres of uncertainty, spanning from software and hardware variability of a single robot to the one associated with mission planning and execution in uncontrollable environments. Hence, MMRSs are facing a number challenges preventing them from being used in our everyday tasks: (i) robots need to be able to operate in unknown environments, (ii) robots will be required to collaborate with each other and even with humans to accomplish complex missions, and (iii) despite failures or malfunctions, robots should never injure people or create loss or severe damage to equipment property. Those challenges are getting targeted from various research perspectives, producing a fragmented research landscape.

In this thesis, we investigate one of the open research questions in this domain, more specifically how to preserve safety while enabling mission completion for MMRSs. We consider that it is fundamental to have a clear separation between safety and mission concerns in order to manage the complexity of the missions in which MMRSs operate. Furthermore, we made an exploration on how to perform on-the-fly adaptation that will enable MMRSs to complete the defined mission while guaranteeing the preservation of safety constraints. We built our research upon a large base of literature of mobile multi-robot systems, self-adaptive systems, and model-driven engineering research.

This thesis proposes a modeling framework for MMRSs that supports specification and execution of missions in MMRSs distinguishing between mission-related and safety-related adaptation mechanisms. It aims to improve the quality of how missions for MMRSs are specified and designed in terms of safety by reducing the intrinsic variability and complexity of todays MMRS missions and promotes reuse of safety-specific mechanisms across missions, projects, and organizations.

Acknowledgements

First, I would like to thank my advisor Patrizio Pelliccione. I truly appreciate that he accepted to take me as a student even under the uncertainty the newly created PhD school posed at the time. I am incredibly grateful to Ivano Malavolta, my (co-)advisor, for his availability, support, insightful ideas and opportunities he has always provided me with.

Special thanks to Gran Sasso Science Institute for giving me this opportunity to pursue doctoral studies. In particular, I am grateful to Rocco De Nicola, the coordinator of the computer science program, for managing to create a compelling working environment during the first years of the institute.

My deepest gratitude goes to David Garlan for hosting me for 11 months at Carnegie Mellon University (CMU) as part of the Fulbright Student Program. I am really thankful to him and his research group (ABLE group) for the knowledge and experience they shared with me. Being at CMU, I had a privilege to meet and have fruitful discussions with some of the most remarkable researchers in the software engineering community. I would like to thank the Fulbright Program for making that possible.

On a more personal note, I am grateful for all people I met during my PhD studies. I have meet people from many different countries and cultures which make big shift in my life perspective. I am particular grateful for meeting great friends like Venkat, Mutti, Yuriy, Tan, Gabriela, Olivera and Abhishek that made my life richer and more enjoyable. To other friends from all around the world, I wish to convey my thanks and my apologies for not being able to mention them personally here.

I would like to profoundly thank Solza Grceva, for her strong encouragement and guidance since I was an undergraduate student. I am also truly thankful to Zoran Gacovski for introducing me to the wonderful world of research and sharing this PhD opportunity with me.

Finally, I would like to thank to the three most important people in my life. I am most thankful to my parents Jovanka and Gorjan and my sister Viktorija for their love and unconditional support during all my life. Their support and encouragement during the PhD process was irreplaceable. Special thanks to my mom Jovanka who always found a way to connect with me and encourage me through difficult times. I dedicate this thesis to them.

Contents

List of Figures

I would like to dedicate this thesis to my loving parents ...

Chapter 1

Introduction

Robotic systems have brought significant socio-economic impacts to human health and quality of life over the past few decades. Variety of definitions in the literature exists defining the term robot. For all of them, the following is in common: a robot is an inteligent device with a certain degree of autonomy, possibly moving within a physical environment, to perform the intended tasks.It is a system that contains sensors, control systems, manipulators, power supplies and software all working together to perform the required tasks [6]. Designing, building, programming and testing robots is a combination of physics, mechanical engineering, electrical engineering, structural engineering, mathematics and software engineering. In some cases natural sciences like biology, medicine and chemistry might also be involved.

In the industry, robots have been widely deployed to do tedious, repetitive, or dangerous tasks, such as assembly, painting, packaging or welding. These preprogrammed robots have been very successful in industrial applications due to their high endurance, speed, and precision in structured factory environments. In a world that is undergoing significant environmental and social change, there will be an increasing demand for robots that leave the safety of their controlled environments and operate in the real world. To extend the functional range of these robots or to deploy them in unstructured environments, robotic technologies are integrated with network technologies to foster the emergence of networked robotics. A networked robotic system refers to a group of robotic devices that are connected via communication network [7]. In a recent book, Rise of the Robots [8], Martin Ford discusses the transition in robotics from special purpose robots, built to operate in highly controlled environments on a specific task, to general purpose robots that can operate in a heterogeneous environment, intermixed with humans, and perform a broad spectrum of tasks. Smart robots equipped with sensors, cameras, and intelligent software promise to bring a new industrial revolution. According to Industrie 4.0 [9], we

are in the middle of the 4th industrial revolution that is based on autonomous and smart Cyber Physical Systems (CPSs) [9], able to cooperate with each other and humans in a safe, autonomous, and reliable manner.

In this thesis we scope our work within the area of multi-robot systems, specifically the mobile multi-robot systems. By Mobile Multi Robot Systems(MMRSs) we refer to a set of mobile robots operating as a team (even together with humans) in a shared environment. A *mobile robot* represents a robotic system consisting of a SW/HW platform carried around by locomotive elements and able to perform tasks in different contexts. The kind of locomotion that the robot is able to perform is primarily decided upon the environment (aquatic, aerial or terrestrial) in which the robot will be operating [10]. Mobility gives robots enhanced operative capabilities. Moreover, it brings additional challenges that increase complexity and tangle variability for these systems. Some of these challenges are common to all mobile robots (e.g. the navigation problem), while others (e.g. walking gait) are peculiar for a specific locomotion type.

Natural catastrophes, nuclear power plant decommissioning, hazardous materials recovery and cleanup, extra-planetary exploration, conflicts, and less dangerous activities, such as delivery services, surveillance, environmental monitoring are just few application scenarios where MMRSs will operate. These scenarios, characterized by repetitive and dangerous tasks, often require a heavy human presence. In order to reduce the human involvement, innovative technologies and approaches represented by mobile robotics are seen as particularly suitable for aiding in the replacement of human beings with robotic systems in those scenarios. In the literature numerous application areas in which MMRSs could be applid are described and discussed. In [11], a categorization of application fields scenarios is represented for Unmanned Aerial Systems (UAS). They are divided into six categories which are presented in Figure 1.1. In these application scenarios, the robots need to reliably perform collaborative tasks beyond their explicitly preprogrammed behavior and quickly adapt to the unstructured and variable nature of tasks [12].

To understand the business potential for MMRSs we did a research on the market. We analyzed a report titled "The Unmanned ground vehicles (UGV) market 2011-2021: Military robots for EOD & COUNTER-IED" (http://tinyurl.com/nf5x3tf) which assesses the business opportunities presented by the critical unmanned systems marketplace. This study examines the 12 leading national markets for unmanned ground vehicles (UGV) by sales, as well as assessing the wide range of factors that are driving sales growth around the world. From the analysis was seen that worldwide governments spend around $702 million on UGVs in 2011. Moreover, according to Washington Post

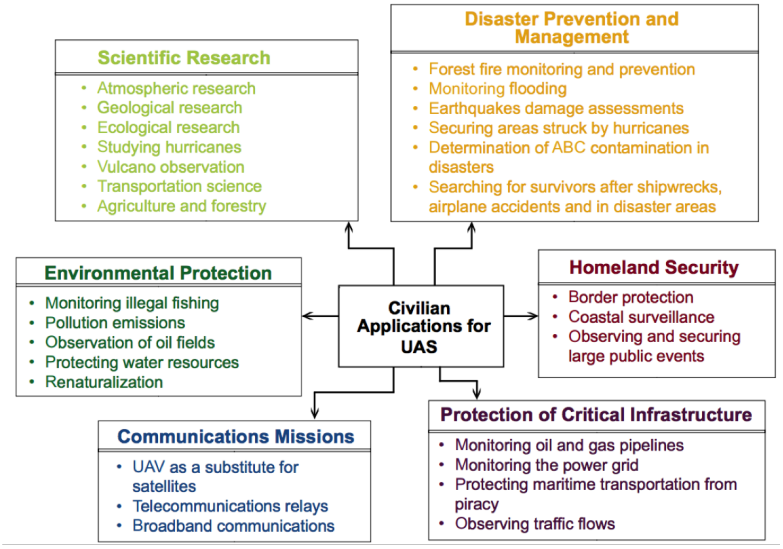


FIGURE 1.1: Application scenarios [11]

(http://wapo.st/1bJueLH) venture investors in the US poured $40.9 million into drone-related start-ups in January-September 2013, more than double the amount for 2012. Furthermore, in the "Unmanned Aerial Vehicle (UAV) Market" research, the UAV market is estimated to be $6,762 million in 2014 and is expected to reach $10,573 million by 2020 (http://tinyurl.com/ooohujg). Moreover, in "Global Underwater Robotics Market 2012-2016", the market of underwater robots is forecasted to grow at a compound annual growth rate (CAGR) of 6.77 percent in the four-year period from 2012 to 2016 (http://tinyurl.com/o78htbq). All in all, the market for industrial robotics is expected to rise at a Compound Annual Growth Rate (CAGR) of 11,5% annually though 2021 and to reach $48.9 billion by 2021, while the total smart robots market is expected to reach USD 7.85 billion by 2020, at an estimated CAGR of 19.22% between 2015 and 2020. All these reports show a clear picture how and why this market will experience continuing growth.

Now, we will present an example of a MMRSs. In the motivating scenario on Figure 1.2 a MMRS is used in a search and recovery scenario. The agents involved in the scenario are the following:

- Unmanned Aerial Vehicles (UAV) - agents that perform a search for a specific location of interest in some environment
- Autonomous Ground Vehicles (AGV) - agents that arrive to the found location of interest and perform some action

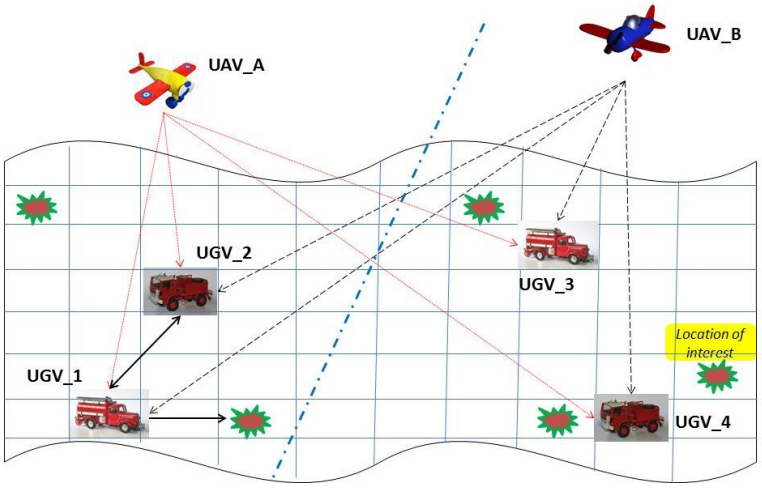


FIGURE 1.2: Examples of mobile robots

Taking in consideration this scenario, we characterize our system as follows: the system consists of heterogeneous agents that have limited resources, partial knowledge of the system and the environment and perform different actions based on their specific knowledge. They operate under dynamic, uncontrollable and partially or fully unknown environments. This introduces a set of uncertainties that are result from incomplete knowledge of the run-time structure of the system (e.g. number of agents performing a particular task in a specific moment) and the environment at design time. Incomplete knowledge of the run-time structure of the system comes from its openness meaning that new entities can join or leave the system at run-time while incomplete knowledge of the environment comes from its dynamics and uncontrollability (e.g., a bird flying in the environment). This means that we need to deal with it when the knowledge becomes available and perform system adaptation. However, existing systems are facing a number of challenges preventing them from being used in our everyday tasks: (i) robots need to be able to operate in unknown environments, (ii) robots will be required to collaborate with each other and even with humans to accomplish complex missions, and (iii) despite failures or malfunctions, robots should never injure people or create loss or severe damage to equipment/property. Those challenges are getting targeted from various research perspectives, producing a fragmented research landscape.

The construction of MMRSs is significantly more challenging than traditional systems due to their mission-criticality (meaning that a loss of resources can lead to possible reduction in mission effectiveness) and their safety-criticality (meaning that a failure or defective design could cause a risk to human life and the environment). Although robotic research has made huge progress in the last decades, the existing solutions seem to be not-yet-ready to be used in everyday life, and in uncontrollable and unknown environments often shared with humans. Commercialisation and adoption of MMRSs in

dynamic environments will only occur if safety aspects are considered and incorporated as first-class concerns in the design of the system. Certification bodies should assure some type of safety certification that relies on a complete understanding of the system. However, for mobile robots that operate in dynamic environments its quite challenging to consider all variants of the overall system. Hence, having the ability to analyse and reason about *safety* independently from the *mission* requires a clear separation of concerns between safety and mission related issues. Mission management and safe behaviour management of multiple robots have been widely recognized and studied in research [13]. However, in order to make MMRSs pervasive and ubiquitous in heterogeneous environments, intermixed with humans, new approaches to guarantee sustainable safety during the mission execution are needed.

In this thesis, we propose a modeling framework that supports specification and execution of missions in MMRSs. We began our research work for this thesis with a general research question that emerges from the aformentioned challenges: **How to preserve safety while guaranteeing (potentially partial) mission completion for MMRSs?**.

In order to identify and evaluate the relevant literature related to our research question, we did a systematic mapping study. A systematic approach gave us an objective, thorough summary and critical analysis of the relevant high-quality literature available on this topic [14, 15]. In the systematic mapping study we aim to identify, classify, and understand existing research on safety in mobile robots rather than just focusing on safety in MMRSs, in order to understand the gaps of the current research on a more general and broader topic.We found that most of the approaches surveyed in this study: (i) focus on a single robot, (ii) do not support adaptive capabilities and (iii) are not able to deal with systems supporting the addition and removal of robots, human actors, etc. at runtime. Furthemore, the majority of approaches in safety for robotic systems lack both rigor and relevance. Therefore, there is the need of new strategies to better support and planning the evaluations of approaches for safety of robotic systems.

To address our main question, we built upon a large base of literature of self-adaptive systems, MMRSs and Model-Driven Engineering (MDE) research. MDE was successfully used and showed positive results in other domains (e.g. avionics, automotive and telecommunications). Because of that, we considered that MDE can be a viable direction for successfully supporting future MMRSs. In order to build the modelling framework, we used Model-Driven Engineering (MDE). Many MDE technologies and concepts (like MOF, OCL, UML, model transformations) are widely known in both research and practice, and have been standardized by international consortia like the Object Management Group (OMG). In the robotics field, having standard technologies is extremely valuable

since it will make less painful to develop, maintain, and reuse software tools and components across different projects and organizations.

As a last part of our work, we evaluated the quality of our proposed contributions both from an experimental and theoretical point of view. In order to validate our generic run-time adaptation in practice we performed an experiment [16]. In this direction, we used the experience acquired from the FLYAQ platform. FLYAQ[1] is a platform for mission planning of autonomous quadrotors.

The main contribution of this thesis is the design of a modeling framework for MMRSs that: (i) supports specification and execution of missions; and (ii) ensures safety while guaranteeing (potentially partial) mission completion.

1.1 Structure of the Thesis

In this section we outline the structure of the subsequent chapters of this thesis:

Chapter 2 gives some background information about MMRSs and their operational context. It discusses research challenges that are identified in designing MMRSs and it scopes the research problem that will be addressed in this thesis.

Chapter 3 provides a complete, comprehensive and replicable picture of the state of the art on safety for mobile robotic systems (MSRs). It discusses a systematic mapping study we conducted to identify, classify, and understand existing research on safety in mobile robots. We focused on safety in mobile robots rather than focusing on safety in MMRSs in order to understand the gaps of the current research on a more general and broader topic.

Chapter 4 discusses the first part of our modeling framework that supports the specification of missions in MMRSs.

Chapter 5 discusses the second part of our modeling framework that supports the execution of MMRSs missions. Furthermore, we present a generic approach for managing collective adaptation of MMRSs in a decentralized fashion at run-time.

In Chapter 6 we discuss an extension of the work done in Chapter 5. In this chapter we present two adaptation resolution methods: one for (potentially partial) resolution of mission problems and one for full resolution of safety problems.

Chapter 7 concludes the thesis by listing our conclusions, the contributions we made and provides research directions for future work.

[1] http://www.flyaq.it/

1.2 Published Material

The research presented in this thesis has resulted in the following peer-reviewed publications:

- Bozhinoski, D., Bucchiarone, A., Malavolta, I., Marconi, A., & Pelliccione, P. (2016, October). **Leveraging Collective Run-time Adaptation for UAV-based Systems.** *In Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on (pp. 214-221). IEEE.* [4]
- Bozhinoski, D., Di Ruscio, D., Malavolta, I., Pelliccione, P., & Tivoli, M. (2015, November). **FLYAQ: Enabling Non-expert Users to Specify and Generate Missions of Autonomous Multicopters.** *In Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on (pp. 801-806). IEEE.*[3]
- Bozhinoski, D., Malavolta, I., Bucchiarone, A., & Marconi, A. (2015, September). **Sustainable Safety in Mobile Multi-robot Systems via Collective Adaptation.** *In 2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems (pp. 172-173). IEEE.*[17]
- Bozhinoski, D. (2015, May). **Managing Safety and Adaptability in Mobile Multi-Robot Systems.** *In Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures (pp. 135-140). ACM.*[1]

The content of Chapters 3 and 6 is pending review in:

- Bozhinoski, D., Ruscio, Di D., Malavolta, I., Pelliccione P., & Crnkovic I. (2017, September). **Safety for Mobile Robotic Systems: a Systematic Mapping Study.** *In Journal of Systems and Software (JSS)* (https://goo.gl/A4bKbD)[2]
- Bozhinoski, D., Garlan D., Malavolta, I., Pelliccione P. (2017, November). **Managing Safety and Mission Completion via Collective Run-time Adaptation.** *In Journal of Systems Architecture: Embedded Software Design (JSA)* (https://goo.gl/kDZYxf)[5]

Chapter 2

Background

2.1 Context

MMRSs have been proposed in the last twenty years in various settings and for different goals, both in research and in practice. Already from their dawn, MMRSs were meant to operate in an uncertain, dynamic context [18]. But context is not the only mutable aspect in the MMRSs ecosystem; variability is multi-dimensional since changes can occur both vertically (within domain and specification of a single robot) as well as horizontally (among the MMRS' parts). In this work, the focus is on the latter – *horizontal variability*.

MMRSs is represented by a set of robots operating as a team (even together with humans) in a shared environment. In such contexts it is becoming more and more profitable the collaboration between humans and the robots. Over the last decades, research in robotics has made huge progress in the fields of image recognition and processing, planning, control, and collaboration. However, we currently have at our disposal a myriad of isolated solutions that are hard to reuse and combine. Software engineering is called to play a key role in securing this new technology's affirmation by making it pervasive and ubiquitous. Powerful methodologies are required to assist the development of robotic software systems, which are expected to be able to:

- operate in unpredictable environments,
- collaborate with other systems for solving problems that could not be solved otherwise neither by one single robot nor by a team of robots belonging to the same category,
- automatically deal with unexpected emergent behaviors that might potentially cause severe misshapes.

To address the increasing complexity and the needs of the variegated nuances of these systems, the robotics and automation industry is working towards the establishment of new international safety standards through the International Organization for Standardization (ISO) for robots and robot systems integration [19]. Establishing the guidelines to regulate a safe use of these innovative technologies is, as a matter of fact, the means to increase their trustworthiness and thereby their appreciation, not only in the research and business sectors, but even in the private social sphere. This is the reason why the necessity for formal verification and validation of safety properties for complex robotic software systems, especially MMRSs, has largely intensified.

In addition, research on MMRSs is dealing with a number of challenges in various domains. Here, we will briefly discuss some of the general challenges that developers, researchers, scientists and practitioners are facing in the different domains.

2.1.1 Control engineering

A control system refers to a collection of components being operated under the direction of some software intelligence called controller [20].

Recently, research has been focusing on safety and efficiency of electro-magnetic, hydraulic, and pneumatic physical actuators and sensors, by trying to address specific concerns such as torque control, mechanical arms trajectory generation and optimization, and under-actuated object manipulation [20].

2.1.2 Environment perception

When considering MMRSs, environment perception refers to the techniques, algorithms, and technologies that a robot can use to get information from the environment it operates in. In this context, challenges refer to the development of efficient sensors for the robots (e.g., position sensors efficiency and accuracy, and light sensors accuracy). Research in the area mainly focuses on object recognition and modeling, site reconstruction and inspection, efficient pose and settings of vision sensors, managing the sensing and environmental uncertainty, and objects tracking and search [21].

2.1.3 Communication and distributed computing

MMRSs require the interaction between numerous heterogeneous components, both hardware and software. This aspect opens a number of research challenges, in terms

of inter-robot communication, distributed computation, dynamic wiring, new technologies integration, and so forth. A large number of robotics middleware frameworks have been proposed [22]. As a representative example, the Robot Operating System (ROS - http://www.ros.org) provides operating system's services such as hardware abstraction, message-passing between processes, and integration with commonly-used robotic libraries (e.g., the Gazebo simulator, the OpenCV vision library). Other robotics middleware frameworks exist, each of which with specific goals and features, such as the Open Robot Control Software (Orocos - http://www.orocos.org) for real-time machine and robot control, the Player (http://playerstage.sourceforge.net) network server for simple and cross-platform interfaces to the sensors and actuators of the robots over the IP network etc.

2.1.4 Tasks and motion planning

Task planning refers to the management of the sequence of high-level actions that allows the robot to accomplish a given mission. For instance, collaborative flying robots have been successfully modeled and implemented as distributed processing systems [23]. Algorithms and frameworks have been proposed for the automatic generation and control of motion plans, with a focus on either trajectory optimization, feasibility, or safe obstacle and trajectories intersection avoidance. A large number of algorithms aim at solving the area coverage problem, each of which with specific benefits, drawbacks, and application domains (e.g., underwater and flying robots with strict flight dynamics) [24].

2.1.5 Reasoning and dynamic adaptation

MMRSs operate in highly dynamic environments where robot failures and environmental changes can never be completely foreseen and pre-programmed, which means that can not be analyzed and checked entirely at design-time. Handling with all the variety and uncertainty of the environment asks for specific reasoning techniques that can provide the MMRSs with the most appropriate adaptation mechanisms at runtime. Research in the area is mainly focused on: learning techniques to detect and effect changes in the environment, building plausible knowledge models, deciding on the relevant forms of representation and manipulation of the information gained through a rich range of contextual factors, integration of the context with the other types of knowledge [25].

2.1.6 Software engineering

From a software engineering point of view, the main challenge is to maximize maintainability, interoperability, and reusability of the robotic system while preserving efficiency, robustness, safety, and reliability [26]. Furthermore, it is very hard to tame the ever-increasing heterogeneity and complexity of multi-robot systems in terms of communication infrastructure, device capabilities and involved stakeholders, while preserving reasonable costs and time to market. The work in this thesis addresses challenges within the software engineering research area. In the next section we will discuss about few classes of software engineering challenges in the domain of Mobile Multi-robot systems (MMRSs).

2.2 Software Engineering Challenges

MMRSs are composed of set of subsystems which behave as a team, thus accomplishing a global mission. These subsystems might vary both in type and in number, during the MMRS' life-time: new robots might come into the picture or current ones might be reconfigured (or self-configured) at run-time as countermeasure to possible faults, malicious attacks or simply due to additional knowledge acquired during the mission execution.

In order to better understand the complexity of MMRSs we can analyse these systems through the aspect of multidimensional variability since changes can occur both vertically (within domain and specification of a single robot) as well as horizontally (among the MMRS parts). While many existing approaches are tackling the challenges emerging out of the vertical variability and provide solutions to problems associated with a single robot, we want to complement those approaches and tackle problems where a single robot can not perform the mission alone because of lack of resources. (e.g., pictures in an open area should be taken, but one robot can't finish the mission because of lack of battery power). As part of [12], we focus only on the of horizontal variability and the challenges that emerge out of it. In this context, we identify two dimensions of horizontal variability as discussed in [12]: system variability, and mission variability.

System variability. Currently robotic software researchers and engineers are mainly focusing on delivering highly efficient implementations of control applications for specific platforms in well-defined operational environments [26]. However, the strive to tackle performance issues has resulted in neglecting other relevant quality attributes of a software system, such as reusability, interoperability, and maintainability [26]. As a

consequence, available solutions, both software and hardware, are incredibly heterogeneous. Additionally, neither software nor hardware has been truly standardized yet, thus making the development of MMRSs involving heterogeneous robots very intricate and somewhat frustrating for developers that find themselves compelled in re-inventing the wheel over and over again. In point of fact, reuse of software components and their integration within a specific robotic application is not properly supported either.

Mission variability. MMRSs are called to operate in diverse usage scenarios and civilian missions, such as damage assessment after earthquakes, telecommunication relays, atmospheric research, and agriculture. Each of these scenarios might require specific quality of services, such as safety, real-time and resource constraints, timing requirements, etc. Moreover, these systems operate in highly dynamic environments: (i) resources that these systems use (energy, memory, connection capabilities, etc.) are limited in capacity, (ii) resources might degrade or even disappear (e.g., wireless connection), (iii) they usually are not extensible during the system's lifetime, (iv) new resources might become available and contextual information typically vary. Quality criteria need to be preserved despite adaptation, which for these systems is the norm rather than the exception. Finally, the variety of missions discussed above implies an extensive assortment of interaction scenarios and involves variegated categories of final users, each with particular characteristics and needs. This calls for diversification in human-robot interfaces, information visualization, suitable abstraction, as well as proper information hiding.

In this section we elucidate the core challenges that we identified as most relevant implications of the two dimensions of horizontal variability. Figure 2.1 depicts the two dimensions for which we discuss a set of research challenges they imply.

2.2.1 Software and hardware variability

In [12], we have identified three main software and hardware variability dimensions that occur when managing the development and adoption of MMRSs.

Platform neutrality. Nowadays, heterogenenity has become a common trait that characterizes robotic software systems, especially when it comes to the hardware composition on which the software system is meant to run. This is due to the fact that specific configurations starring any combination of CPUs, GPUs and FPGAs have been proposed to serve the individual system's distinctive purpose, such as processing huge amount of data, at high-rate, and in real-time. This heterogeneity, together with the lack of a standardized platform-agnosticity of robotic software solutions, makes cross-platform development intractable at the moment. That is the reason why a lift in the

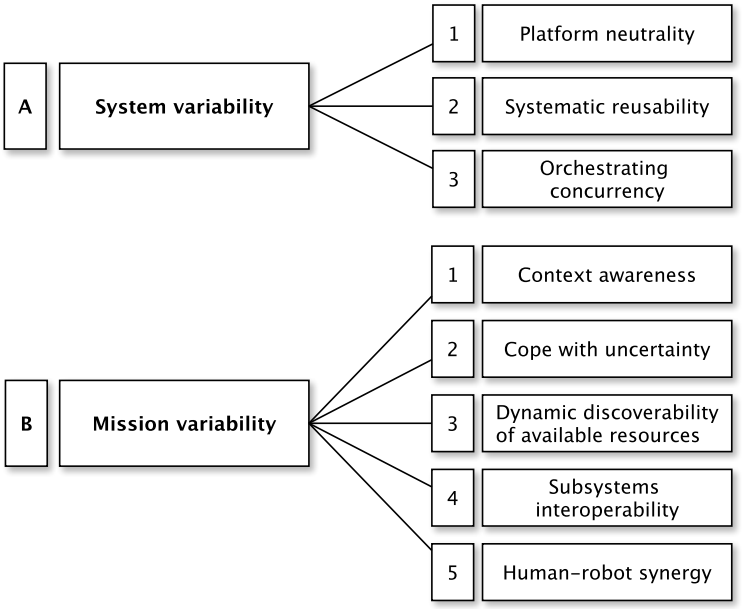


FIGURE 2.1: Variability dimensions and challenges

level of abstraction at which robotic software is developed towards platform neutrality. Such a lift cannot disregard an investigation of a basic common infrastructure to assist the development of robotic software, user interfaces and management of diverse application domains is strongly needed. The ideal goal would be the standardization of basic systems functions, scalability to different robots and platform variants, transferability throughout the network, integration from multiple suppliers, maintainability throughout the robot life-cycle and software updates and upgrades over the robot's lifetime.

Systematic reusability. Robotic software developers often experience a sense of frustration when they have to develop from scratch a "new" application, even if very similar to previous releases for different projects, because they do not have at their disposal appropriate support for capturing and exploiting commonalities [26]. In our investigation, we encountered divergent implementations of the same functionality in MMRSs such as perception, navigation, planner, controller etc. Usually, the implementation of a specific functionality is initiated by domain experts that design the system and take key decisions on algorithms that then the software developer implements in terms of code. The lack of systematic, disciplined, and quantifiable robotic software engineering methodology, as well as comprehensive abstraction mechanisms for handling the increasing complexity of robotic software systems, lead to countless, similar, but not congruent, isolated solutions which cannot be easily reused and combined. Thus, a solid collaboration between different working groups in robotic systems is hard to establish.

For this reason, tackling this challenge is of paramount importance in order to make development of robotic software sustainable on the market where expectations for new generation robotic devices is growing at incredible pace. Systematic reusability calls also for appropriate integration means (e.g., architectures, connectors, integration patterns) that ease collaboration and integration of existing solutions while still satisfying the system's purpose under a controlled degree of uncertainty. These approaches should be efficiently performed at run-time and should be able to guarantee different quality attributes both in isolation and, more importantly, in combination since they usually heavily affect each other. Therefore, specific effort shall be devoted to trade-off analysis of quality attributes for enabling optimization with guarantees.

Orchestrating concurrency. MMRSs are meant to be loosely coupled and concurrent. This means that the global state of the system might be suddenly changed by actions of the individual robots that operate independently of each other. The inherent concurrence typical of MMRSs makes their development and execution extremely challenging. The number of robots collaborating in the MMRSs can be arbitrary, and each of them can be programmed, re-programmed, or can even automatically adapt to contextual information in an independent manner with respect to the other robots composing the MMRS. Moreover, MMRSs may expose emerging properties that represent unexpected behaviours that stem from interactions between the system parts and with the context [27]. Emergent properties might be beneficial, but they can be also harmful, e.g., if they compromise system's safety. For these reasons, suitable orchestration mechanisms shall be provided to suitably control concurrency of MMRSs.

2.2.2 Mission planning and execution variability

When planning and executing missions involving multi-robot systems there are different variability aspects that have to be taken into account as discussed in the remaining of the section.

Context awareness. The development of context-aware applications is always a challenging and complex task. These applications adapt to changing context information, where context might be physical, computational, and user-related. Modelling real scenarios require processing of context facts and reasoning upon them to attain a form of context information that is appropriate for the different missions. Moreover, context information is usually collected from different sources that differ in quality and that are often failure prone [28]. To minimize the possibility to run into failing situations at run-time derived by faulty context information, development and representation of the context for each mission should be supported by adequate modelling and reasoning

techniques. These techniques need to address the variegation of context information types and the relationships among them, but also to cope with the uncertainty that comes along with context information. Tackling these issues would contribute in reducing complexity and improving maintainability and evolvability of the MMRS. Overall, the context modeling plays a key role in order to be able to properly deduce the actions that the robots shall perform in order to execute the missions by meeting the environmental constraints.

Cope with uncertainty. On the one hand MMRSs are required to live under uncertainty: they are intrinsically dynamic and might evolve according to available resources, unpredictable contextual information, etc. On the other hand, they are subject to rigid development practices imposed by certifications and compliance to standards that are inescapable. Therefore, in order to ensure the quality users expect, the evolution of MMRSs should be controlled by suitable mechanisms able to ensure that defective and malicious adaptations do not affect the entire MMRS, especially when it comes to safety and reliability, but without leaving behind other essential aspects such as security, performance and dependability. To be applicable in practice, these techniques should be lightweight and able to abstract from formal details that are not meant to be overseen by the end-users.

Dynamic discoverability of available resources. MMRSs operate in highly dynamic environments where uncertainty and unforseen changes can unanticipatedly occur at any time. Matching the needs of the mission with the available resources right away is one of the basic and key aspects when designing MMRSs. In order for this to happen, we have to look for suitable robots satisfying a given set of constraints, and for which we have the access permission in the particular moment. In highly dynamic environments, new or recovered robots can show up in the environment at any time and, in order for the MMRSs to exploit them, a mechanism that handles dynamic discoverability of available resources and constraints is pivotal and it is particularly crucial in the case of new, unknown robots entering the MMRS. Thereby, the system should be able to communicate and recognize the robots' characteristics and to do the necessary adjustments to mission plans and strategies accordingly.

Subsystems interoperability. MMRSs are characterized by the integration of heterogeneous subsystems that collaborate towards a common goal. Managing heterogeneity of the individuals forming the MMRSs is a challenging task due to the nature of the subsystems that differ in resources, protocols, platforms etc. In order for the MMRSs to provide appropriate coordination between heterogeneous subsystems, flexibility has to be enforced in many different aspects such the ability to make individuals using different communication protocols to communicate, the possibility to easily modify a robot's

task, as well as the capability of plug and play both applications and individuals in the running MMRS.

Human-robot synergy. Human-robot synergy has been delineated by Goodrich et al. [29] as the problem of understanding and shaping the interactions between one or more humans and one or more robots. Since robotic software systems are created and used to do work for and with humans, these interactions permeate in all of robotics, from design to deployment, from execution to maintainance, and even in the autonomous branch. For this reason it is of cardinal importance to invest effort in evaluating capabilities of both humans and robots, and designing proper means of interaction. This multidisciplinary task involves diverse natural as well as engineering fields, such as linguistics, psychology, mathematics, computer science, and design. We share Goodrich and Schultzs claim that, to properly address this task, the robotic software designer should take into account the five main characteristics that affect interaction that are: (i) level of robot autonomy, (ii) type of information exchange, (iii) MMRS composition (in terms of individual subsystems), (iv) learning of humans and robots in the MMRS, and (v) mission (or task) definition and configuration.

2.3 Research Problem

In the previous sections we captured the current research issues and state-of-the-art challenges in MMRSs development and mission design. This area has recently received increasing attention from the robotics community due to the fact that software development efforts are mainly focused on delivering highly efficient implementations while neglecting other quality attributes like maintainability, safety, interoperability etc.

This thesis aims to fill the current gap in terms of approaches for MMRSs that address the problem of proper adaptation for *mission* and *safety* related properties. One of the most important reason for success of industrial robots is the assurance of appropriate level of safety [30]. We use the following definition of **safety**: *safety represents the absence of catastrophic consequences on the user(s) and the environment* [31]. However, industrial safety standards are focused on safety by isolating the robot away from people [19]. The new technological advancements in robotics enable robots to move from isolated environments to more unstructured and dynamic environments where they operate among people performing collaborative tasks beyond their explicitly pre-programmed behaviour. Hence, it is fundamental for safety to be reconsidered and greatly enhanced at this point of time.

To address the increasing complexity and the needs of the variegated nuances of mobile robots, the robotics and automation industry is working towards the establishment of new international safety standards through the International Organization for Standardization (ISO) for robots and robot systems integration [19]. Commercialisation and adoption of mobile robots in dynamic environments will only occur if the safety aspects are considered and incorporated as first class elements in the design of the system. Establishing the guidelines to regulate a safe use of these innovative technologies is, as a matter of fact, the means to increase their trustworthiness and thereby their appreciation and use, not only in the research and business sectors, but even in the private social sphere. Certification bodies should assure some type of safety certification that relies on a complete understanding of the system. However, for mobile robots that operate in dynamic environments its quite challenging to consider all variants of the overall system due to their adaptive behaviour [11]. Adaptability is the extent to which a system adapts to changes in its environment [32]. An adaptable system can tolerate changes in its environment without external intervention. Whenever the system's context changes, the system has to decide whether it needs to adapt. We consider context as any information which is computationally accessible and upon which behavioural variations depend. Actors (entities that interact with the system), the environment(the part of the external world with which the system interacts), and the system itself may contribute to the context change.

Traditionally in the development of systems there is a clear distinction between design activities and runtime execution. The more critical the system is the more of the choices will be made at design-time in order to reduce the reconfiguration options to a set of predictable configurations. MMRSs are safety-critical and mission-critical systems meaning they should be intensively validated at design-time while at run-time, they have a predictable behavior, time and resource consumption. However, there is a growing application need for more flexible systems which will be able to operate in unpredictable environments, coping with unanticipated situations, while still being able to ensure safety properties and perform the mission. To ensure mission and safety properties, all variants of the system have to be checked against all possible threats configurations. In highly adaptive systems as MMMRSs, the number of system variants usually grows exponentially since the context in which these systems have to operate are often unpredictable and unknown. Hence, handling uncertainty upfront is often infeasible (or expensive).

Given the problem description we raised the following research question:

How to preserve safety while guaranteeing (potentially partial) mission completion for MMRSs?

In order to approach this question we identified two sub-research questions that we plan to address:

R1. How to create a clear separation of concerns in modeling safety and mission aspects for MMRSs?

Distinguishing between safety-related and mission-related aspects is of most importance when modeling complex missions for MMRSs systems as the nature of mission objectives is different to the safety objectives. Safety is a first class concern in our missions as robots can collaborate with humans to accomplish the mission. In this context, the system should always satisfy all safety objectives, while the mission can be partially satisfied. Hence, managing complex missions requires a clear separation of concerns between safety and mission aspects. This way an operator can focus on the mission functional specification, while a safety engineer can only focus on safety-specific mechanisms that are generic and independent from the functional behaviour of the system. In this thesis, we made an exploration of suitable ways for modeling safety and mission aspects at design-time and run-time.

R.2.How to guarantee safety and (potentially partial) mission completion for MMRSs in relation to operational context changes at run-time?

The spectrum of mission in which MMRSs operate, requires them to be both mission-critical and safety-critical systems. The definition of missions at design-time include only the information that is available at that time. This means we need to deal with context changes when the knowledge about it becomes available. To address this issue, a proper management of the run-time phase is required. In this thesis, we make clear distinction about which decisions and analysis should be made at design-time versus decisions at run-time in order for the system to be able to complete the mission and to guarantee safety. An adaptation should be performed on the fly every time an unexpected system or environmental feature is observed. However, there are few major challenges in performing adaptation on-the-fly. One challenge is to deal with the question of which part of the system should be engaged in an adaptation. This is not trivial at all, since solutions for the same problem may be generated at different levels. For instance, an issue of a robot (i.e., a drop of the battery level of a UAV below a safety threshold) can be resolved in the scope of its mission, by re-calculating its navigation plan, or in the wider scope with the engagement of other robots and supporting systems (e.g., a UAV trajectory manager). The challenge here is to understand these levels and create a mechanism, which decides the right scope for an adaptation for a given issue. The other challenge is to understand how multiple entities in a collective adaptation can adapt altogether and transactionally and what type of negotiation must take place to decide the right behavioural changes that should be applied on each side.

2.4 Solution Overview

Mission management and safe behaviour management of multiple robots has been widely recognized and studied in research [13]. However, in order to make MMRSs pervasive and ubiquitous in heterogeneous environments, intermixed with humans, new approaches to guarantee sustainable safety while mission execution are needed. The problems described above are result of the early stage of development in which this area of research currently is. For that reason, in this thesis we did investigation in this area and propose suitable ways for modeling and analysis of safety and mission properties at MMRSs at design-time and run-time. We built our research upon a large base of literature of mobile-multi robot systems, self-adaptive systems and model-driven engineering research. In particular, by means of MDE it is possible to systematically concentrate on different levels of MMRS abstractions at which all involved stakeholders can operate [33] for (i) improving the quality of MMRSs in terms of safety (ii) reducing the intrinsic variability and complexity of today's MMRSs, and (iii) promoting the reuse of software and hardware components across MMRSs.

The goal of the thesis is to provide guidelines and principles for researchers and developers on how to model and analyze a mission specification for MMRSs at design-time and run-time.

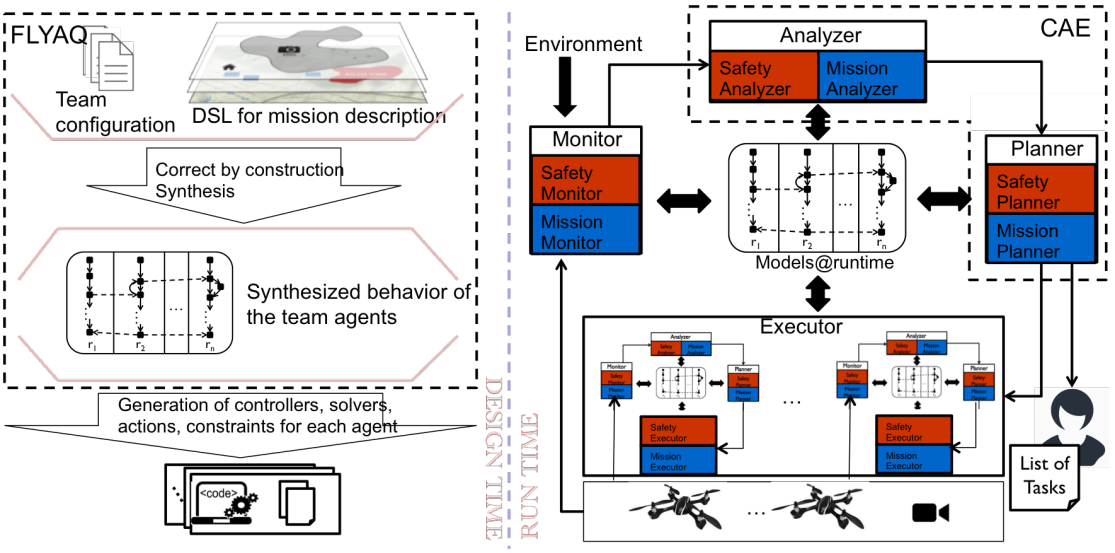


FIGURE 2.2: General overview of our modeling framework

The main contribution of the thesis is a modelling framework that supports specification and execution of missions in MMRSs based on a methodology which explicitly takes into account separation of concerns between safety and mission related problems. The work proposed here aims at providing contributions towards assurance of safety properties

for MMRSs while guaranteeing (potentially partial) mission completition in a dynamic context.

Figure 2.2 presents our modeling framework. The left side shows the first part of our modeling framework that supports the specification of missions in MMRSs as described in Chapter 4, while the right side through the MAPE-K loop illustrates the second part of our modeling framework that supports the execution of MMRSs missions (Chapter 5, Chapter 6). More details about the framework will be given in the corresponding chapters.

Chapter 3

State of the Art

3.1 Introduction

In this chapter we present a study which **goal** is to identify, classify, and evaluate the state of the art on safety for MRSs in terms of its publication and research trends, technical characteristics, and potential for industrial adoption. Achieving such a goal enabled us to draw research challenges and implications for future research on safety for MRSs.

In order to target our goal, we apply a well-established methodology from the medical and Software Engineering research communities called **systematic mapping** [34, 35]. The aim of a systematic mapping study is to provide an unbiased, objective and systematic approach to answer a set of research questions about the state of the art and research gaps on a given topic. A mapping study follows well-defined and replicable principled process for both the search and selection of relevant studies, and the collected data and results synthesis tend to be more quantitative and qualitative [16, § 4.4]. Through our systematic mapping process, we selected 58 primary studies among 1,274 potentially relevant studies fitting at best three research questions we identified (see Section 3.3.1). Then, we defined a classification framework composed of more than 50 different parameters for comparing state-of-the-art approaches, and we applied it to the 58 selected studies. Finally, we analysed and discussed the obtained data for each parameter of the classification framework and how it fits in the research landscape about safety for MRSs.

The **main contributions** of this study are:

- a reusable *comparison framework* for understanding, classifying, and comparing methods or techniques for safety for MRSs;

- a *systematic review* of current methods or techniques for safety for MRSs, useful for both researchers and practitioners;
- a discussion of *emerging research challenges and implications* for future research on safety for MRSs;
- a complete *replication package* of this study containing detailed reports, raw data, and analysis scripts for enabling independent replication and verification of this study.

To the best of our knowledge, this work presents the first systematic investigation into the state of the art on safety for MRSs. The results of this study provide a complete, comprehensive and replicable picture of the state of the art of research on safety for MRSs, helping researchers and practitioners in finding characteristics, limitations, and gaps of current research on safety for mobile robotic systems.

The **main findings** produced by our study are summarized in Table 3.1. By presenting and discussing the above mentioned results we are the first to provide an overview of the state of the art about safety for MRSs, forming a solid foundation for future research on safety for MRSs.

TABLE 3.1: Main results and implications of the study

Results	**Implications**
R1) **Single vs Multi-robots**: most of the studies surveyed in this paper focus on a single robot.	I1) There is the need of solutions addressing safety when multiple robots need to collaborate with each other in order to accomplish complex missions.
R2) **Openness and capability to cope with uncertainty**: many of the surveyed studies do not support adaptiveness capabilities and most of them are not able to deal with open systems, i.e., systems supporting the addition and removal of robots, human actors, etc. at runtime.	I2) The adoption of MRSs in tasks of everyday life would require more investigation in adaptiveness capabilities as well as in dealing with open systems.
R3) **Compliance to standards**: many domain-specific standards related to safety are currently available. Only a minority of approaches are compliant to standards that specifically target safety aspects.	I3) When developing a robotic system, specific standards have to be taken into account to make it compliant to them and safe for the considered application domain.
R4) **Rigor and Industrial Relevance**: the majority of evaluations in safety for robotic systems lack both rigor and relevance.	I4) New strategies are needed to ensure an adequate rigor and relevance when planning the evaluation of approaches for safety of robotic systems.
R5) **Research community on software engineering and robotics**: even though there is a growing interest, the community of software engineering for robotic systems is still not consolidated.	I5) The challenge for the research community is to promote a shift towards well-defined engineering approaches able to stimulate component supply-chains and significantly impact the robotics marketplace.

Chapter outline. The chapter is organized as follows. In Section 3.2 we provide background notions for setting the context of our study by clarifying and discussing (i)

mobile robotic systems, (ii) safety for mobile robotic systems, and (iii) existing studies on safety for MRSs. Section 3.3 describes in details the research methodology we followed for designing, conducting, and documenting the study, followed by a description of the obtained results in Section 3.4. Obtained results are discussed in Section 3.5. We present limitations and threats to validity in Section 3.6. Related works are discussed in Section 3.7, whereas Section 3.8 closes the chapter.

3.2 Background

3.2.1 Mobile Robotic Systems

Robots have been successfully deployed in industry to improve productivity and perform dangerous, tedious or repetitive tasks [30]. In the literature, a variety of definitions exists defining the term "robot" [6, 36, 37]. All of them share the following concept: *a robot is an intelligent device with a certain degree of autonomy that contains sensors, control systems, manipulators, power supplies and software all working together to perform the required tasks.* Under this perspective, a **mobile robot** represents a robotic system consisting of a SW/HW platform carried around by locomotive elements and able to perform tasks in different contexts. The kind of locomotion that the robot is able to perform is primarily decided upon the environment (aquatic, aerial or terrestrial) in which the robot will be operating [10]. Mobility gives robots enhanced operative capabilities, but at the same time increases complexity and brings additional safety challenges.

In order to reduce the human involvement in scenarios that are characterized by repetitive and dangerous tasks (eg. natural catastrophes, nuclear power plant decommissioning, extra-planetary exploration, or less dangerous activities, such as delivery services, surveillance, and environmental monitoring), innovative technologies and approaches represented by mobile robotics are seen as particularly suitable for aiding in the process of replacement of the human beings with robotic systems. That will lead to a society where mobile robots will operate in a dynamic environment and perform the necessary tasks in these scenarios. But, if we want mobile robots to be widely accepted and adopted among the general public, it is fundamental to carefully consider safety aspects.

3.2.2 Safety for MRSs

One of the most important reasons for the success of industrial robotics is its assurance of a high degree of safety. However, industrial safety standards are focused on safety

by isolating the robot away from people [19]. The new technological advancements in robotics enable robots to move from isolated environments to more unstructured and dynamic environments where they operate among people performing collaborative tasks beyond their explicitly preprogrammed behaviour. Hence, it is fundamental for safety aspects to be reconsidered and greatly enhanced at this point of time. In these terms, we use the following definition of safety: *safety represents the absence of catastrophic consequences on the user(s) and the environment* [31].

To address the increasing complexity and the needs of the variegated nuances of mobile robots, the robotics and automation industry are working towards the establishment of new international safety standards through the International Organization for Standardization (ISO) for robots and robot systems integration [19]. Commercialisation and adoption of mobile robots in dynamic environments will only occur if the safety aspects are considered and incorporated as first class elements in the design of the system. Establishing the guidelines to regulate a safe use of these innovative technologies is the means to increase their trustworthiness and thereby their appreciation and use, not only in the research and business sectors, but also in the private social sphere. Certification bodies should assure some type of safety certification that relies on a complete understanding of the system. However, for mobile robots that operate in dynamic environments it is quite challenging to consider all variants of the overall system due to their adaptive behaviour [11]. Recently, researchers have put their focus on the potential for using robots to aid humans outside strictly industrial environments, in more unstructured and dynamic ones [38]. Because of that, it is strongly recommended to revise safety properties for MRSs.

Finally, as of today we did not find any evidence that could help us in assessing the impact of existing research on *safety in mobile robots*. With this study we aim at helping researchers and practitioners in identifying the characteristics, challenges, and gaps of current research on this topic, its future potential, and its applicability in practice in the context of real-world robotic projects.

3.3 Study Design

Figure 3.1 shows the overview of the process we followed for carrying out this study. The overall process can be divided into three main phases, which are the classical ones for systematic mapping studies [16, 35]: planning, conducting, and documenting. In the following we will go through each phase of the process, highlithing its main activities and produced artifacts.

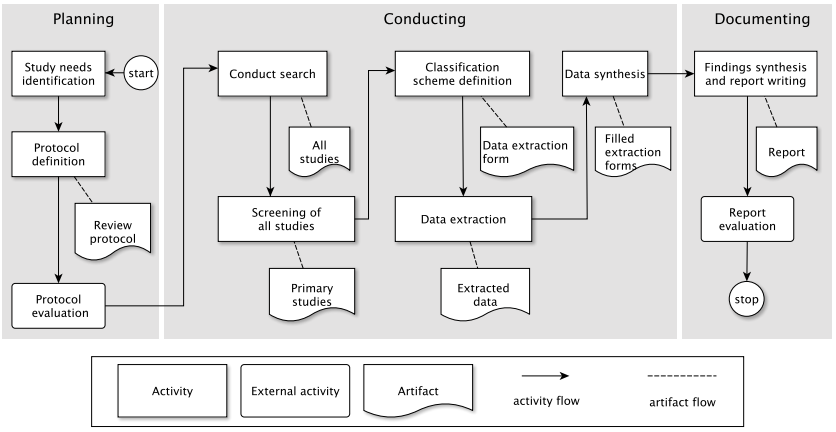


FIGURE 3.1: Overview of the whole mapping process

Planning. It is the first phase of our study and it aims at (i) establishing the need for performing a mapping study on safety for MRSs (see Section 3.7), (ii) identifying the main research questions (see Section 3.3.1), and (iii) defining the review protocol detailing each step of the whole study. The output of the planning phase is a well-defined review protocol. In order to mitigate potential threats to validity, our review protocol has been circulated to external experts for independent review and we refined it according to their feedback[1].

Conducting. In this phase we carried out each step of the above mentioned review protocol. More specifically, we performed the following activities:

- *Conduct search*: in this activity we applied a search string to well-known academic search databases (see Section 3.3.2). The output of this activity is a comprehensive list of all the candidate studies resulting from the search.
- *Screening of all studies*: candidate entries has been filtered in order to obtain the final list of primary studies to be considered in later activities of the study. The basis for the selection of primary studies is the inclusion and exclusion criteria described in Section 3.3.2.
- *Classification framework definition*: we created a classification framework to compare the selected primary studies. The classification framework has been designed so to collecting the information needed for answering the research questions of this study [16]. This acticity will be described in more details in Section 3.3.3.
- *Data extraction*: in this activity we analysed each primary study, and we filled the data extraction form with the extracted information. Filled forms have been

[1]We thank Richard Torkar (University of Gothenburh, Sweden) and Wasif Afzal (Mälardalen University, Västerås, Sweden) for their precious feedback on the review protocol.

collected and aggregated in order to be ready to be analyzed during the next activity. More details about this activity will be presented in Section 3.3.4.

- *Data synthesis*: this activity focussed on a comprehensive summary and analysis of the data extracted in the previous activity. The main goal of this activity is to elaborate on the extracted data in order to address each research question of our research. The details about this activity are in Section 3.3.5.

Documenting. The main activities performed in this phase consist of (i) a thorough elaboration on the data extracted in the previous phase with the main aim of setting the obtained results in their context, (ii) the analysis of possible threats to validity, specially the ones identified during the definition of the review protocol (in this activity also new threats to validity may emerge), and (iii) the writing of a final report describing in details the design and results of this research.

3.3.1 Goal and Research Questions

We formulate the goal of this research by using the Goal-Question-Metric perspectives (i.e., purpose, issue, object, viewpoint [39]). Table 5.1 shows the result of the above mentioned formulation.

Table 3.2: Goal of this research

Purpose	Analyse
Issue	the publication and research trends, characteristics, and potential for industrial adoption
Object	of existing approaches for safety for MRSs
Viewpoint	from a researcher's and practitioner's point of view.

The goal presented above can be refined into the following main research questions.

- **RQ1**: *What are publication and research trends of the studies about safety for mobile robotic systems?* Objective: to *identify* and *classify* publication trends, targeted venues, and the main contributions of researchers in safety for mobile robotic systems over the years.

- **RQ2**: *How do existing approaches address safety for MRSs?* Objective: to *identify* and *classify* existing approaches for safety in MRSs in order to build (i) a solid foundation for classifying existing (and future) research on safety for MRSs and (ii) an understanding of current research gaps in the field of safety for MRSs.

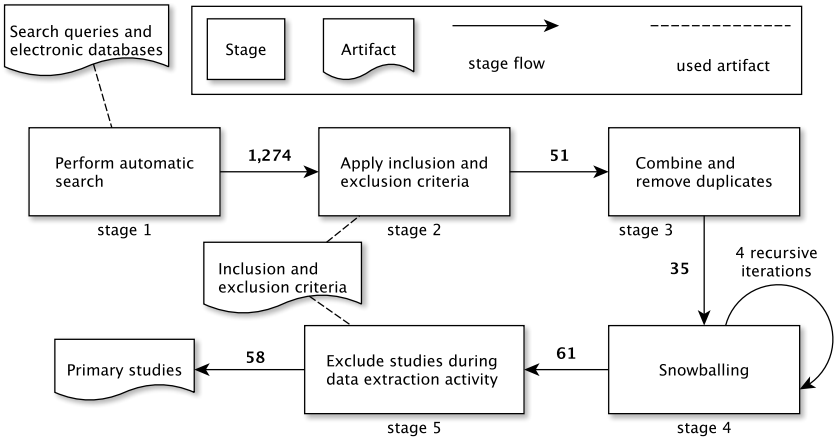


FIGURE 3.2: The search and selection process of this research

- **RQ3**: *What is the potential for industrial adoption of existing approaches for safety for MRSs?* Objective: to *assess* how and if the current state of the art on safety for MRSs is ready to be transferred and adopted in industry. Here we consider criteria such as the rigor and precision of the applied validation strategies (e.g., in-the-lab experiment, industrial application), the realism and scale of the performed evaluations, etc.

Answering those research questions will provide a solid foundation for understanding the state of the art on safety for MRSs, together with its research gaps and future challenges. The above listed research questions will drive the whole systematic review methodology, with a special influence on the primary studies search process, the data extraction process, and the data analysis process.

3.3.2 Search and Selection

The success of any systematic study is deeply rooted in the achievement of a good trade-off between (i) the coverage of existing research on the topic and (ii) having a manageable number of studies to be analysed [34, 35]. In order to achieve the above mentioned trade-off, our search strategy consists of two complementary methods: automatic search and snowballing. As shown in Figure 3.2, we designed our search strategy as a multi-stage process in order to have full control on the number and characteristics of the studies being either selected or excluded during the various stages. In the following we give a brief description of each stage of our search and selection process.

Stage 1. In this stage we performed automatic searches on electronic databases. In order to cover as much as possible relevant literature, four of the largest and most complete scientific databases were chosen as the sources of primary studies of this stage, namely:

IEEE Xplore Digital Library, ACM Digital Library, SpringerLink, and ScienceDirect. The selection of these electronic databases is guided by (i) their high accessibility, (ii) their ability to export search results to well-defined, computation-amenable formats, and (iii) because they have been recognized as being an efficient means to conduct systematic literature reviews in software engineering [40, 41].

To create the search string, we break down our research questions into individual facets (population, intervention, comparison, outcomes, context - PICOC) as discussed in [42]. In our study, the PICOC elements that we identified are as follows:

- **Population:** mobile robotic systems;
- **Intervention:** approaches that address safety in mobile robotic systems;
- **Comparison:** not applicable;
- **Outcomes:** the classification framework populated with the identified primary studies;
- **Context:** academic peer-reviewed publications with a software engineering perspective.

Then we draw up a list of synonyms, abbreviations, and alternative spellings, which combined by logical ANDs and ORs gave the search string. The obtained search string is given below and it has been tested by executing pilot searches on IEEE Xplore Digital Library.

> (mobile **OR** ground **OR** water **OR** fly* **OR** sail* **OR** unmanned **OR** self **OR** autonomous) **AND** (robot* **OR** vehicle*) **AND** (safe* **OR** fault **OR** failure) **AND** software

For the sake of consistency, the search strings has been applied to an identical set of metadata values (i.e., title, abstract and keywords) from all electronic databases. This stage resulted in a total number of 1,274 potentially relevant studies.

Stage 2. The main goal of this stage is to consider all the selected studies and filter them according to a set of well-defined inclusion and exclusion criteria. As suggested in [35], we decided the selection criteria of this study during its protocol definition, so to reduce the likelihood of bias. In the following we provide inclusion and exclusion criteria of our study. In this context, a study will be selected as a primary study if it will satisfy *all* inclusion criteria, and it will be discarded if it will met *any* exclusion criterion.

I1) Studies proposing an approach for safety for an MRS[2].

I2) Studies focussing on safety in MRSs from a software engineering perspective (e.g., no control theory or mechatronics studies, no studies focussing on hardware, etc.).

I3) Studies providing some kind of evaluation of the proposed methodology (e.g., via a case study, a survey, experiment, exploitation in industry, formal analysis, example usage).

I4) Studies subject to peer review [16] (e.g., journal papers, papers published as part of conference proceedings will be considered, whereas white papers will be discarded).

I5) Studies written in English language and available in full-text.

E1) Studies *exclusively* focussing on safety for industrial and other immobile robots.

E2) Secondary studies (e.g., systematic literature reviews, surveys) [16].

E3) Studies in the form of tutorial papers, short papers, poster papers, editorials, because they do not provide enough information.

In this stage, it is fundamental to select papers objectively. To this end, as suggested by [16], two researchers independently assessed a random sample of studies, then the inter-researcher agreement has been measured using the Cohen Kappa statistic; we obtained a Cohen Kappa statistic of 0.80, which is a good indication of the objectiveness of the performed selection. This stage resulted in a total number of 51 relevant studies.

Stage 3. In this stage all studies from the first stage have been combined together into a single set. Duplicated entries have been identified and merged by matching them by title, authors, year, and venue of publication. This stage resulted in a total number of 35 studies.

Stage 4. The main goal of this stage is to enlarge the set of relevant studies by considering each study selected in the previous stages, and focussing on those papers cited by it. More technically, we performed a *closed recursive backward and forward snowballing* activity [44]. From a practical point of view, we went through each selected study and we included also the relevant studies either cited by or citing it (based on Google Scholar [44]). The inclusion of the additionally considered papers into the set of primary studies was based on our inclusion/exclusion criteria. This stage largely increased the number of potentially relevant studies, bringing it to 61. As a possible explanation of

[2]In the context of this research an *approach* can be considered as an organized set of methods and techniques, possibly supported by a tool [43].

this fact, we noticed that researchers used a very heterogeneous terminology when writing the title, abstract, and keywords of their studies; this fact may negatively impact our automatic search, which may have missed some potentially relevant studies. We included the snowballing activity in order to mitigate this potential threat to validity. As a further confirmation, the study reported in [45] empirically observed that similar patterns and conclusions are identified when using automatic search and snowballing, especially when they are used in combination.

Stage 5. This stage has been performed in parallel with the data extraction activity. Basically, the idea is that when reading a study in details for extracting its information, researchers could recognize that it was out of scope, and so it has been excluded. This stage led us to the finalized set of 58 primary studies of our research, which is comprised of 58 entries.

3.3.3 Classification Framework Definition

In our study, the **classification framework** is composed of three facets, each of them addressing a specific research question (see Section 3.3.1) and discussed below.

3.3.3.1 Publication and research trends (RQ1)

In the following we list all the data we collected for analysing publication and research trends on safety for MRS:

- *publication year*: for extracting the publication tendency per year;
- *targeted venue*: analysed in terms of their distribution, and also with respect to their distribution over time. These are the types of publication venues we considered: journal paper, book chapter, conference paper, workshop paper.
- *main research contributions*: we will identify the distribution of primary studies with respect to their main contribution. Categories for research contributions are derived from [46] and can include values such as “method”, “architecture”, “tool”; they are discussed in details in Section 3.4.1.1.
- *application field independence*: while piloting this study we noticed that in the discussion of related work of some papers authors were referring to both domain-specific approaches and generic ones. Based on this, we decided to categorize our primary studies about whether they are independent with respect to any application field (e.g., abstract approaches orthogonal to any application field) or

not (e.g., approaches that are specifically tailored to self-driving cars, agriculture, environmental monitoring).

3.3.3.2 How safety for MRSs is managed (RQ2)

Since research question RQ2 is at the core of our research, the creation of its corresponding facet in the classification framework demands a detailed analysis of the contents of each primary study. In light of this, we followed a systematic process called *keywording* [46] for building this facet of our classification framework. Keywording aims at reducing the time needed in developing a classification framework and ensures that it takes the considered studies into account [46].

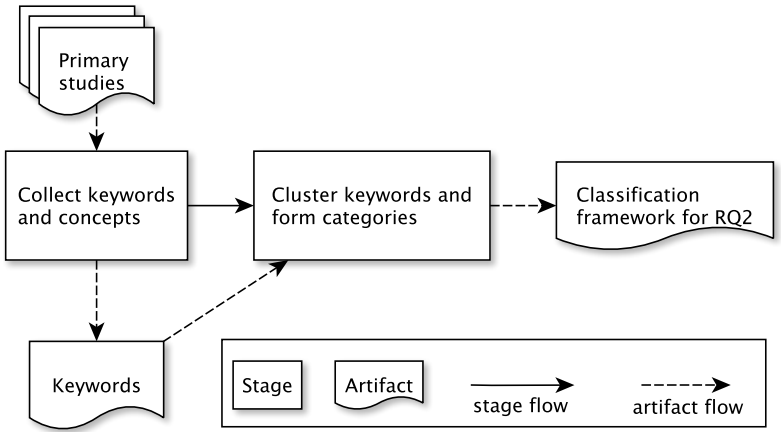


FIGURE 3.3: Overview of the keywording process

As shown in Figure 3.3, keywording is done in two steps:

1. *Collect keywords and concepts*: we collected keywords and concepts by reading the abstract of each primary study. When all primary studies have been analysed, all keywords and concepts have been combined together to clearly identify the context, nature, and contribution of the approach. As suggested in [46], when the abstract of a primary study was not informative enough, then we analysed also its introduction and conclusion sections. The output of this stage is the set of keywords extracted from the primary studies.

2. *Cluster keywords and form categories*: when keywords and concepts have been finalized, then we performed a clustering operation on them in order to have a set of representative clusters of keywords. The output of this stage is the classification framework containing all the identified clusters, each of them representing a specific aspect of safety for MRSs. The specific categories emerging from the keywording process are described in Section 3.4.2.

3.3.3.3 Potential for industrial adoption (RQ3)

For answering RQ3 we considered the following parameters:

- *application field*: the application field where the proposed approach has been evaluated (e.g., exploration missions, service robotics, self-driving vehicles);
- *applied research method*: here we distinguished between approaches validated in a controlled setting (or in the lab) and approaches evaluated in real-world (industrial) contexts;
- *validation/evaluation strategies*: here we extracted the strategies applied for assessing the proposed approaches (e.g., real deployment, simulation-based, proof of concept), independently of whether they are performed in the context of validation or evaluation research;
- *technology readiness level (TRL)*: it has been proposed by the Horizon 2020 European Commission for the 2014/2015 work program[3], the TRL is a metric for measuring the maturity of a given technology;
- *rigor and industrial relevance*: we measured the precision, exactness and realism of the evaluation of each primary study by applying the rigor and industrial relevance metrics proposed by Ivarsson and Gorschek [47];
- *industry involvement*: whether each primary study has been carried out only by academics, practitioners (or a mix therof) for understanding how researchers and practitioners collaborate on safety for MRSs.

3.3.4 Data extraction

As already said, the classification framework is the base of the data extraction form, i.e., a well-structured form to store the data extracted from each primary study. For each of these studies, we collected in a spreadsheet a record with the extracted information for subsequent analysis. As suggested in [16], the data extraction form (and thus also the classification framework) has been independently piloted on a sample of primary studies by two researchers, and iteratively refined accordingly. Once the data extraction form was setup, we considered each primary study and its corresponding data extraction form has been filled with the extracted data.

[3] http://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf

In order to validate our data extraction strategy, we performed a sensitivity analysis involving two researchers on 10 randomly selected primary studies to check whether the results were consistent independently from the researcher performing the extraction. In this context, each disagreement has been discussed and resolved, with the intervention of a third researcher, when necessary.

3.3.5 Data Synthesis

This activity involved collating and summarising the data extracted from the primary studies[35, § 6.5] with the main goal of producing the actual map of current research on safety for MRSs. When possible, in this research we applied both quantitative and qualitative analysis methods, depending on the nature of each specific parameter of the classification framework.

For each parameter of the classification framework we divided our *quantitative* analysis on two main steps: (i) we counted the number of primary studies falling in relevant categories in the context of the specific parameter and (ii) we aggregated and visualized the extracted information to better clarify similarities and differences between the primary studies.

For what concerns the analysis of *qualitative* data, we used the already presented keywording method for identifying also the possible values of each parameter of the classification framework, and then we analysed and summarized the trends and collected information in a quantitative manner.

Finally, we carried out a narrative synthesis of the results obtained both quantitatively and qualitatively; this step allowed us to (i) perform an evidence-based interpretation of the main findings coming from the previous analyses and (ii) extract the main challenges and implications for future research.

3.4 Results

In this section we present the results of this study for answering our research questions (see Section 3.3.1). For each parameter of our classification framework we report both quantitative data and an interpretation of the obtained results.

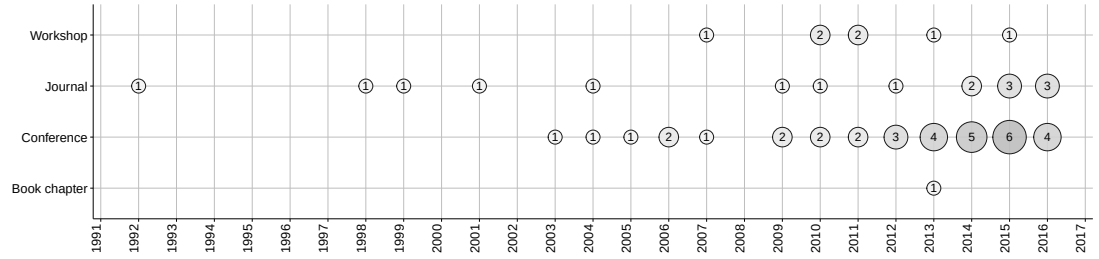


FIGURE 3.4: Distribution of primary studies over the years - results

3.4.1 Publication and research trends (RQ1)

This research considers a set of 58 primary studies, each of them published in different years and venues. Figure 3.4 shows the distribution of the primary studies over the years and by the type of venue where they have been published[4]. The obtained data clearly shows a growing trend in terms of **publication intensity**, with most of the studies published in the very recent years; specifically, 46 studies over 58 have been published from 2009 to 2016 (with an average of more than 5 publications per year), where 17 studies have been published only in 2015 and 2016. If we look at the publication numbers before 2009 we have a drop to less than one publication per year. These results are a confirmation of the growing scientific interest on safety for mobile robotic systems, specially in the last years.

More on a historical perspective, the first study on safety for mobile robotic systems (P11) has been published in the Applied Intelligence international journal in 1992. In P11 the authors proposed an automated diagnostic method for keeping an autonomous underwater vehicle operational for several weeks without human intervention. The approach was based on a distributed fault-tolerant control system aiming at managing unpredicted faults by preserving its overall performance level. The approach makes the assumption that the normal behaviour of each component is available at design time.

We also classified the primary studies by (i) type of publication and (ii) targeted publication venues. As shown in Figure 3.4, the most common **publication type** is conference paper (34/58), followed by journal papers (16/58), workshop papers (7/58), and finally book chapters (1/58). The high number of journal and conference publications may be seen as an indication of the good level of quality of research on safety for mobile robots, specially in the last years.

In Table 3.3 we report the **publication venues** that hosted more than two publications (the last row of the table is an aggregate of all the publication venues with two or

[4]Our search activity covers the research studies published until January 2017, thus we potentially have only partial data for 2016.

TABLE 3.3: Targeted publication venues

Venue Acronym	**#Studies**	**Studies**
Intelligent Robots and Systems (IROS)	6	P1, P5, P20, P22, P32, P58
International Conference on Robotics and Automation (ICRA)	4	P6, P36, P40, P54
International conference on Automated Software Engineering (ASE)	3	P14, P21, P55
IEEE Transactions on Robotics and Automation (TRA)	2	P2, P7
Robotics and Autonomous Systems (Journal)	2	P10, P41
International Journal of Robotics Research (IJRR)	2	P19, P35
Conference Towards Autonomous Robotic Systems (TAROS)	2	P43, P45
IEEE Conference on Emerging Technologies and Factory Automation (ETFA)	2	P18, P56
IEEE International Symposium on High-Assurance Systems Engineering (HASE)	2	P21, P55
International Conference on Advanced Robotics (ICAR)	2	P24, P37
International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)	2	P12, P27
Others	32	P28, P11, P26, P17, P8, P46, P31, P25, P47, P50, P9, P29, P49, P3, P34, P44, P39, P33, P53, P23, P16, P4, P13, P30, P57, P48, P15, P55, P38, P51, P52, P42

less publications). What strikes the eye is the extreme fragmentation of the targeted publication venues (43 unique venues for 58 publications). Nevertheless, we can observe that the most targeted venues (i.e., the ones targeted by at least two primary studies, see Table 3.3) are quite homogenous and dedicated to robotics, autonomous systems, automation, and high-assurance systems. Given their focus on aspects related to safety for MRSs, we can consider those venues as good candidates for future publications on this area.

3.4.1.1 Research contributions

In order to characterize where researchers are focussing their efforts, we extracted the main research contribution of each primary study. Categories of research contributions are derived from [46], and can be one or more of the following alternatives: model, method, architecture, metric, tool.

The results of our analysis are shown in Figure 3.5.a. It does not come as a surprise that the main contribution of the majority of primary studies is a *method* to address specific concerns about safety for MRSs (43/58); this result does not come as a surprise since our inclusion criterion I1 is explicitly dealing with studies proposing either a method or a technique for safety. The second most recurrent research contribution is *architecture* (21/58); those studies present the fundamental concepts or properties related to the

safety of an MRS by reasoning on its elements, relationships, and in the principles of its design and evolution [48]. This result is interesting since it confirms that safety has been treated as a system-level property by researchers, and that considering safety at a higher level of abstraction is a valuable and effective strategy for attacking the problem. Other studies contribute with the information, representations, and abstractions for safety of MRSs (*model*, 11/58), and developed tools or prototypes for safety of MRS (*tools*, 9/58). As a final consideration, no primary study has as main contribution *metrics*, indexes, or measures to assess certain properties of safety of MRSs. By following old adage that *what gets measured gets managed*, working on safety-specific metrics for MRSs can be an added value for the field and surely an interesting research gap to be filled by future research.

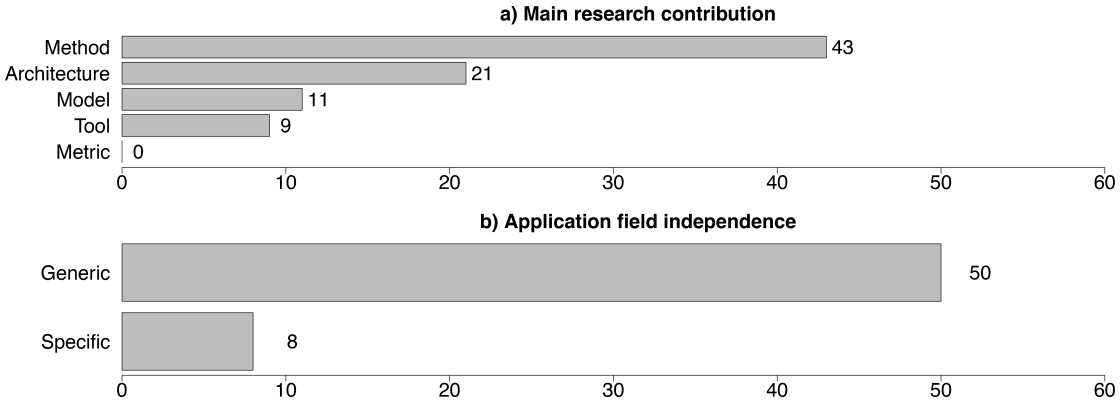


FIGURE 3.5: Types of research contribution (a) and application field independence (b) - results

3.4.1.2 Application fields independence

As shown in Figure 3.5.b, almost all the primary studies are *generic* with respect to any application field. This means that those studies are kind of orthogonal and can be applied to some extent to different types of robots, tasks to be performed, operational contexts, etc. For example, the authors of P9 achieved genericity by applying the well-known abstraction and automation principles of the Model-Driven Engineering paradigm (MDE, [49]). By quoting their own words, their approach *directly enables an implementation-independent reuse of the safety-related part of a robot controller between different releases, since the RuBaSS declaration does not need to change when the underlying software changes (except that names shared between RuBaSS rules and component interfaces must be kept consistent). Moreover, the infrastructure can be reused in a range of products: the code generator can be directly reused whereas low-level interfaces to sensors and actuators will be specific to each robot. Safety-related customisation for the products is thus mainly achieved at the higher level, using the safety language* (P9).

Application-specific approaches have been proposed in 8 primary studies (namely, P7, P9, P11, P39, P42, P50, P54, and P56), with application fields ranging from health to domestic or industrial robotics.

Highlights - Publication trends (RQ1): The first primary study on safety for MRSs dates back to 1996. While the number of studies published in the 1992-2008 time frame has been quite limited, a growing interest on the topic can be noticed from 2009 onwards. Most of the papers have been published in journals and conferences, across a large number of different venues. The majority of the primary studies propose new (mainly generic) methods for achieving safety for MRSs.

3.4.2 How safety is managed (RQ2)

This section aims at identifying and classifying existing methodologies that address safety in mobile robotic systems. In order to refine our investigation, this research question has been decomposed into more detailed subquestions. Therefore, we discuss about:

- *safety management*: how the proposed approach considers safety-related aspects (e.g., specific mechanisms for safety, the level of abstraction, whether safety is treated as first class element of the approach or not, etc.);
- *system characteristics*: the features of the systems supported by the proposed approach (e.g., cooperative versus local adaptation, the type of robots, their cardinality, etc.);
- *models*: it is about the models of the system and their features (i.e., whether the proposed approach is based on model-based techniques, the purposes of the used models);
- *standards*: the standards to which the proposed approach is compliant (e.g., IEC61508, ISO10218);
- *hazards*: about the characteristics of the hazards managed by the approach (i.e., whether they are unexpected, their scope and cardinality).

3.4.2.1 Safety management - World knowledge

As shown in Figure 3.6.a, most of the approaches (45/58) rely on a local knowledge of the environment. This means that the knowledge about the environment (including

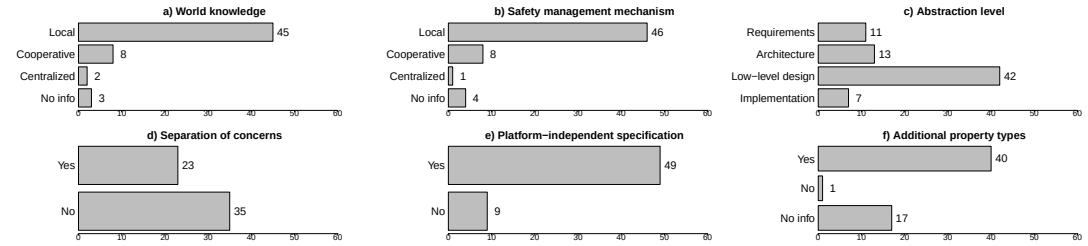


FIGURE 3.6: Safety management - results

other robots involved in the mission) is local to each robot, without mechanisms to share knowledge between different robots. 2 approaches have a centralized world knowledge, meaning that the knowledge of the overall system is maintained by a centralized entity. 8 approaches have cooperative world knowledge and this means that there are mechanisms to share knowledge between different robots that take part in the mission.

It is important to note that only two approaches with local knowledge involve multi-robots, namely P43 and P51. This explains why we have a majority of approaches that rely on local knowledge. In general, we might say that having a centralized world knowledge in multi-robot systems might hamper the adoption of decentralized algorithms for (re)planning, issues resolution, and so on.

3.4.2.2 Safety management - Mechanism

As shown in Figure 3.6.b, most of the approaches (46/58) adopt local safety mechanisms, i.e. safety mechanisms that are conceived to work on single robots, without any cooperation. As highlighted also above (Section 3.4.2.1), most of these approaches focus on single robots, and therefore this result is expected. The exceptions are P43 and P51 that deal with multiple robots even though they have local safety mechanisms, and P54 that has both local and centralized safety mechanisms. Centralized safety management mechanism means that there exist an entity managing the safety aspect of the overall system. As can be seen in the figure only 1 approach has a centralized safety management mechanism. Instead, 8 approaches rely on cooperative safety mechanisms, meaning that safety mechanisms involve a cooperation between different robots. Finally, 4 approaches provide no information about this aspect.

3.4.2.3 Safety management - Abstraction levels

As shown in Figure 3.6.c, the abstraction level of the safety management spans from requirement till implementation. A requirements value means that safety is considered when eliciting/specifying the requirements of the system (e.g., generic safety rules

written in a non-technical way). Architecture means that safety is considered at the architectural level (e.g., they talk about architectural tactics, styles, architectural patterns, system infrastructure, communication topology, etc.). Low-level design means that safety is considered at the design level (e.g., design patterns, design models, etc.). Finally, implementation means that safety is considered at the source code, programming level.

The majority of the approaches works at the design level and the minority at the implementation level. Moreover, as shown in Table 3.4, only 3 approaches exclusively address safety at the implementation level, while. among the 7 approaches addressing safety at the implementation level, other 3 approaches address safety also at the design level, and another one at the architecture level.

This testifies that it might be difficult to manage safety directly at the implementation level and it is more profitable to deal with it at more abstract levels. On the other side of the medal, it might be impracticable to address safety exclusively at the requirement level, in fact, as shown in Table 3.4, the majority of approaches touching the requirement level, are also dealing with other levels (specifically, 8 out of 11 approaches that focus on the requirement level are also managing safety at the design level). The design level seems to be the most appropriate level, followed by the architecture level.

TABLE 3.4: Safety management - Abstraction levels

Level(s)	**Number of studies**
Requirements	3
Requirements + Low-level design	8
Architecture	9
Architecture + Low-level design	3
Architecture + Implementation	1
Low-level design	28
Low-level design + Implementation	3
Implementation	3

3.4.2.4 Safety management - Separation of concerns

As shown in Figure 3.6.d, for the majority of the approaches (35/58), the management of safety-specific issues (e.g., safety rules) is not kept separated from the functional management of the robots (e.g., the mission). Keeping a separation of concerns means for instance that the approach prescribes a special layer for managing safety, which is totally separated from the rest of the system.

TABLE 3.5: Additional property types (as reported in the primary studies)

Property	#Studies	Studies
Performance	13	P1, P2, P4, P10, P11, P13, P18, P23, P27, P28, P31, P34, P58
Functional correctness	12	P17, P41, P48, P49, P50, P51, P52, P53, P54, P55, P56, P57
Reliability	7	P18, P30, P31, P37, P38, P40, P46
Dependability	5	P10, P14, P20, P24, P31
Usability	5	P16, P21, P22, P27, P32
Robustness	4	P24, P35, P36, P37
Availability	3	P10, P22, P35
Effectiveness	3	P1, P11, P35
Reusability	3	P16, P27, P32
Efficiency	3	P1, P2, P10
Modularity	2	P16, P27
No additional type	1	P3
Integrability	1	P21
Validity	1	P21
Applicability	1	P21
Maintainability	1	P45
Complexity	1	P45
Flexibility	1	P45
Expressiveness	1	P45
Upgradeability	1	P16
Reusability	1	P27
Repeatability	1	P32
Security	1	P22

3.4.2.5 Safety management - Platform independent specification

As shown in Figure 3.6.e, for the high majority of the approaches (49/58), the specification of safety-specific aspects (e.g., safety constraints, properties, rules, invariants specifying assumptions about hardware) is independent from the underlying platform (e.g., ROS, hardware, operating system, etc.). This is a good characteristic of the platform since this can enable reusability of software modules across various platforms.

3.4.2.6 Safety management - Additional property types

As shown in Figure 3.6.f, most of the approaches deal with properties that are different from safety. In fact, 40 approaches deal with additional properties, only one approach is exclusively focused on safety, P3, and 17 papers do not provide information. Table 3.5 shows the additional properties and adds a reference to primary studies that are addressing the specific properties. There is a big variety of additional properties that are addressed by the primary studies - 22 different additional properties considered by

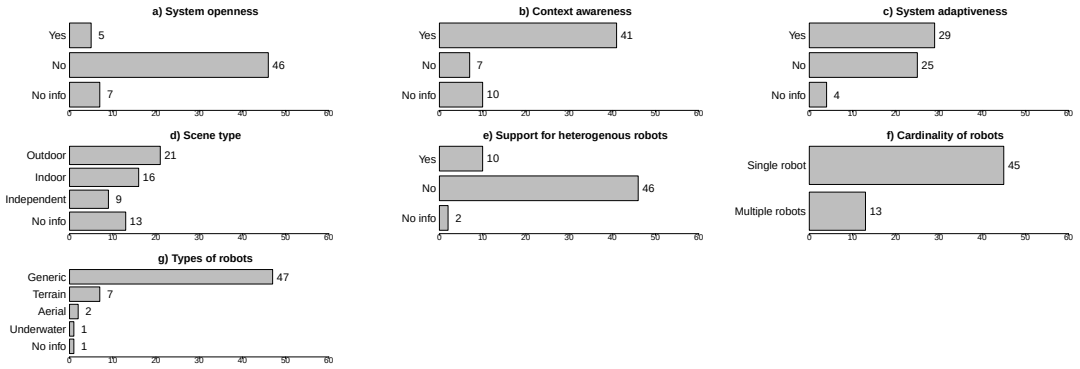


FIGURE 3.7: System characteristics - results

the 40 primary studies that consider additional properties. Performance is the most addressed property, followed by functional correctness.

3.4.2.7 System characteristics - Openness

As shown in Figure 3.7.a, most of the approaches are unable to deal with open systems (only 5 approaches, namely P2, P22, P48, P49, P53, are able to deal with open systems). By open systems, we mean systems that can accept external entities at run-time (e.g., new robots, new human actors). This implies that most of the approaches that have been proposed are not able to manage safety once the system evolves in terms of addition or removal of robots and/or other types of agents, including humans. This is indeed an interesting research direction since systems of the near future will be necessarily characterised by openness, and it is often impossible to assess at design time the exact boundaries and topology of the system.

3.4.2.8 System characteristics - Context awareness

As can be seen in Figure 3.7.b, most of the approaches (41/58) deal with systems (including the robots) that are able to understand some key properties about the operational context of the robots (e.g., presence of obstacles, existence of other robots, etc.). 10 out of 58 approaches do not provide information. Context awareness is another important characteristic to enable the adoption of robots in real life scenarios, where often the operational environment is (partially) unknown and uncontrollable.

3.4.2.9 System characteristics - Adaptiveness

Figure 3.7.c shows that 29/58 approaches have adaptiveness capabilities, meaning that the system (including the robots) is able to adapt (e.g., behaviour adaptation, trajectory recalculation, goal renegotiation, etc.) in order to find a solution depending on

some change in the context. 25/58 approaches do not support this functionality, and 4 approaches provide no information. Adaptiveness might be considered in conjunction with context awareness since awareness of the context is a required capability in order to support adaptiveness.

3.4.2.10 System characteristics - Scene type

Figure 3.7.d describes the ability of the system to work indoor (21/58), outdoor (16/58), or independent of the scene (9/58). Some approaches provide no information in this concern (13/58). Please notice that we categorised an approach as independent only if the approach explicitly mentions about its independence ability.

3.4.2.11 System characteristics - Heterogeneous robots

Another peculiar system characteristic is the capability of managing teams consisting of robots of different types (e.g., robots for grabbing objects, for video streaming, sensing and discovering relevant information). According to Figure 3.7.e most of the analyzed systems (46/58) do not have the capability of managing heterogeneous robots. Only 10 systems provide users with such a functionality, whereas 2 analyzed systems do not provide a clear statement about that.

3.4.2.12 System characteristics - Cardinality of robots

Missions can be executed by one or more robots. Indeed the management of different robots introduce additional challenges mainly related to their collaboration and coordination. As shown in Figure 3.7.f most of the analyzed systems (45/58) support missions performed by a single robot (e.g., self-driving car), while few of them deal with the management of multiple robots.

3.4.2.13 System characteristics - Type of robots

In order to manage different kinds of missions it is preferable that the used system provides users with functionalities that are robot independent. According to the performed analysis, 7 out of 58 analysed systems are specific to terrain robots (see Figure 3.7.g), 2 specifically conceived for aerial robots, and 1 for underwater robots. Most of the system are generic (47/58) and paper P40 does not provide any details about the supported robot types.

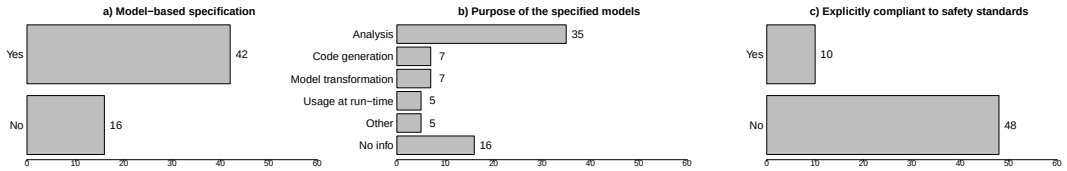


FIGURE 3.8: Model-based specifications and standards - results

3.4.2.14 System characteristics - Platform

Another aspect characterizing robotic systems is related to the platform used for their implementation. While performing the analysis, we counted 17 different platforms in addition to ad-hoc ones. In Table 3.6 we show the most used platforms (at least two occurrences). ROS is one of the most used platforms (13/58), even though the majority of the analyzed primary studies propose their ad-hoc technologies (20/58). Such numbers are justified by the need of abstraction layers taming the complexity of writing software for robotic systems. Even though ROS was explicitly designed with such a goal, ad-hoc platforms are also employed e.g., to overcome limitations of ROS (e.g., scalability and reliability) that might be critical for some application domains.

TABLE 3.6: Platform used by the implementations of the approaches

Platform	**#Studies**	**Studies**
Ad-hoc platform	20	P6, P7, P9, P10, P13, P14, P15, P16, P19, P20, P21, P22, P25, P26, P32, P34, P42, P43, P44, P47
ROS	13	P6, P12, P17, P29, P30, P31, P32, P37, P38, P45, P51, P54, P56
OPROS	3	P29, P30, P38
ADE	2	P36, P37
Corba	2	P9, P23
OpenPRS	2	P8, P31
OrocosRTT	2	P27, P58
RTAI	2	P10, P16

3.4.2.15 Models - Model-based specification

Engineering mobile robotic systems has to take into account several aspects that might go from requirement elicitation to the specification of hardware characteristics. Consequently, the adoption of model-based techniques can help developers in managing the different aspects by increasing abstraction and enabling automation. Many approaches make use of models (42/58 as shown in Figure 3.8.a) for various purposes, e.g., to support the specification of missions, safety constraints, hardware invariants, etc. Only 10 approaches do not make use of models for developing and using the robotic systems at hand.

3.4.2.16 Models - Purpose of the specified models

By continuing the discussion related to the previous point, the adoption of models can be done for different purposes. Most of the considered approaches (35/42 as shown in Figure 3.8.b) adopt models for analysis purposes (e.g., feasibility assessment, mission execution time prediction, etc.). Some of them (7/42) use models for generating the code of the modeled systems or to apply model-to-model transformations (7/42) targeting models that are in the form, which is more convenient for the particular analysis task. Some of the analyzed systems (5/42) use models at run-time e.g., to support the execution of the mission while it is executed. The papers in the *Other* category are P15, P18, P29, P37, P41. In P15 models are used to support the run-time and dynamic adaptation of systems due to unforeseen environment changes. Adaptive systems are considered also in P18 and P29 that propose the adoption of models to deal with fault tolerant aspects of the systems being developed. Fault management is also the main topic of P37, which adopts models for specifying systems consisting of multiple mobile robots. P41 proposes the adoption of models for supporting the development of autonomous systems, which have to be self-healing.

3.4.2.17 Standards - Compliant standards

Mobile robotic systems are very complex as testified also by the number of standards that are considered when developing them (see Table 3.7). According to Figure 3.8.c 10/58 approaches are compliant to standards that specifically target safety aspects. As shown in Table 3.7, each approach can adopt more than one standard depending on the peculiar aspects of the system being developed. For instance, P35 and P42 make use of 4 standards each. The former, proposes an approach to develop safe control systems and as a such it refers to the following standards:

- IEC61508 – Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems;
- ISO10218 – Robots and robotic devices - Safety requirements for industrial robots;
- ISO13855 – Safety of machinery - Positioning of safeguards with respect to the approach speeds of parts of the human body;
- ANSI/RIA R15.06 – Industrial Robots and Robot Systems - Safety Requirements.

In P42 authors propose an approach to verify the correctness of vision pipelines in agricultural settings with the aim of improving the safety of the systems being developed. The proposed approach considers the following standards:

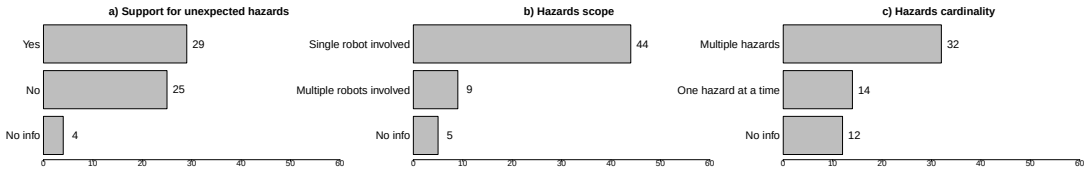


FIGURE 3.9: Hazards - results

- ISO13482 – Robots and robotic devices - Safety requirements for personal care robots;
- ISO25119 – Tractors and machinery for agriculture and forestry Safety-related parts of control systems;
- ISO18497 – Agricultural machinery and tractors – Safety of highly automated agricultural machines;
- IEC61496 – Safety of machinery - Electro-sensitive protective equipment.

As it is possible to notice, the standards that are referred by the existing approaches vary much depend on the particular application domains where the considered robotic systems will be employed.

TABLE 3.7: Standards with compliant approaches

Standard	**#Studies**	**Studies**
IEC61508	4	P5, P21, P35, P52
ISO13482	3	P33, P39, P42
ISO10218	1	P35
ISO13855	1	P35
ANSI/RIA R15.06	1	P35
ISO14121	1	P21
ISO11199	1	P21
ISO12100	1	P39
IEC60204	1	P39
ISO25119	1	P42
ISO18497	1	P42
IEC61496	1	P42
ISO62262	1	P9
IEC61608	1	P9
OASIS	1	P49
RTCADO178C	1	P45

3.4.2.18 Hazards - Unexpected environment hazards

In order to employ mobile robotic systems in real contexts, it is important that they have the capability of reacting to unexpected environment threats, such as the presence of unpredicted obstacles, the presence of humans in the operating area, etc. As shown in

Figure 3.9.a, the majority of the analyzed systems (29/58) implement such a capability. The primary studies P3, P4, P8, and P38 do not give explicit information about that. In particular, P3 proposes an approach to support the diagnosis of complex systems. P4 discusses all the concepts that have to be taken into account when designing autonomous systems by touching different peculiar aspects like communication, control, and navigation. The focus of P8 is supporting testing activities when developing the control software for autonomous systems. With the aim of improving the quality of the software of robotic systems, P38 proposes an approach to manage faults of components based on the OPRoS platform.

3.4.2.19 Hazards - Scope

When considering unexpected environment hazards, systems can be distinguished with respect to their capability of managing threats impacting or due to a single robot (44/58 as according to Figure 3.9.b), from those occurring because of the cooperation and coordination of different robots. Only 9 out of 58 analyzed systems are able to manage unexpected hazards coming from multi-robots systems.

3.4.2.20 Hazards - Cardinality

Another level of complexity related to the management of unexpected environment hazards is related to the capability of the system to manage one or multiple threats at a time. According to the performed analyzed and as shown in Figure 3.9.c, most of the analyzed systems are able to deal with multiple hazards, whereas 14 out of 58 have the capability of managing only one hazard at a time. Unfortunately, 12 primary studies do not provide explicit information about such characteristic.

Highlights - Management of safety of mobile robotic systems (RQ2): The vast majority of the primary studies manage safety by relying on knowledge which is: (i) local to each robot and (ii) exploited to implement local safety mechanisms without any cooperation with other robots.

Safety is considered at different levels of abstraction, by spanning from requirement specification till implementation, even though most of the approaches work at design level by making use of different kinds of models. Safety-specific concerns are typically specified in a platform- and robot-independent manner. Contrariwise, the actual management of safety is not kept separated from the functional management of robots.

Most of the primary studies do not seem to address safety in case of different kinds of robots and of dynamic additions or removals of robots and/or other agents. Context awareness is instead implemented by the vast majority of the analysed studies, which are able to sense some key properties of the considered operational context of robots, and consequently to implement adaptiveness capabilities in case of context changes.

Few primary studies are able to manage safety for multi-robot systems and the majority of the analysed approaches work atop of ad-hoc platforms, even though ROS is gaining more and more momentum.

Further research is still needed to overcome important limitations of MRSs, in particular the capability of reacting to unexpected environment hazards by still keeping safety under control.

3.4.3 Potential for industrial adoption (RQ3)

In this section we will discuss the results on how existing research on safety for mobile robotic systems can be potentially adopted in real industrial projects.

3.4.3.1 Application field

Mobile robotic systems is a wide domain with many specific fields, such as exploration missions, service robotics, self-driving vehicles. As already discussed in Section 3.4.1, totally different approaches can be applied for solving concerns that are specific for each application field. In order to provide guidance to researchers and practitioners on which application fields have been concretely investigated by researchers, in Table 3.8 we report the application fields which have been considered during the evaluation of the proposed approaches. Practitioners can consider this table as an indication of research approaches that can be potentially applied in real-world projects in specific application fields.

As shown in the table, exploration mission is the most recurrent application field (11/58). For example, the P6 a model-driven approach based on a domain-specific language called RobotML has been applied on autonomous motions of mobile robots in unknown structured environment. Service robotics are also quite recurrent (11/58), indeed safety is a fundamental quality of this kind of robotic systems due to their potentially continuous interaction with human users. As an example, in P35 a robot has the task to (i) retrieve objects from a collection area, (ii) sort them into good or bad based on quality, (iii)

TABLE 3.8: Recurrent application fields

Application fields	#Studies	Studies
Exploration mission	11	P6, P8, P18, P19, P22, P24, P27, P28, P34, P43, P53
Service robotics	11	P2, P10, P16, P32, P33, P34, P35, P39, P47, P50, P53
Not specified	9	P4, P11, P12, P44, P46, P48, P49, P51, P55
Search and rescue	6	P1, P5, P33, P34, P36, P41
Navigation tasks	4	P30, P37, P38, P58
Self-driving vehicles	4	P13, P14, P15, P26
Medical care	3	P7, P9, P21
Playing soccer (RoboCup)	2	P3, P23
Industrial robotics	2	P54, P57
Automatic cleaning	1	P25
Scientific research	1	P31
Transportation	1	P52
Waste cleanup	1	P2
Drawing lines of soccer fields	1	P56
Environment protection	1	P17

identify the location of the human user and stop at a safe distance facing them - in this way the user can hold an object for inspection in front of the robot, (iv) negotiate with the human, and (v) fully take hold of the object for putting it into the collection area. Indeed, during the described interaction the robot must keep a safe operating speed and distance from the human.

It comes as a surprise that 9 primary studies out of 58 did not take advantage of any application field when evaluating the proposed approaches; in those cases the evaluation has been kept at an abstract level. Those studies may suffer from the industrial adoption point of view because a practitioner may be reluctant to use the proposed approach since there is no evidence about how it can be applied in practice (not even with an example).

A number of other application fields have been used in research on safety for mobile robotic systems; they are reported in the lower part of Table 3.8, e.g., search and rescue (6/48), navigation tasks (4/48), self-driving vehicles (4/48). Also, a number of very peculiar application fields have been applied in only one primary study; they are: automatic cleaning (P25), scientific research (P31), transportation (P52), waste cleanup (P2), drawing lines of soccer fields (P56), and environment protection (P17).

3.4.3.2 Applied research method

As discussed in [34] and [50], from a high-level perspective a research solution can be assessed by means of two main research methods: *validation* and *evaluation*. Concretely, *validation* focuses on specific properties of the proposed solution and it is done in a controlled setting or in the lab; *evaluation* aims at investigating on the new situation brought by the proposed solution and it takes place in real-world (industrial) contexts.

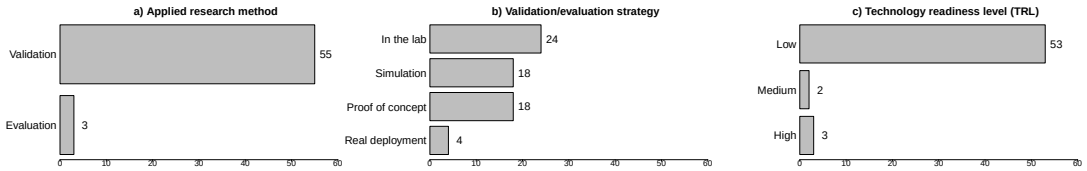


FIGURE 3.10: Applied research method (a), validation/evaluation strategy (b), and technology readiness level (c) - results

In the context of this study, evaluation potentially provides a higher level of evidence about the practical applicability of a proposed approach for safety of mobile robotic systems.

As shown in Figure 3.10.a, the vast majority of our primary studies provides only a *validation* of the proposed approach (55/58). This result is a clear call for researchers on safety for mobile robotic system for assessing their approaches on real-world industrial contexts, potentially leading to a smoother technology transfer of their proposed research. As a starting point for achieving this result we can get inspired by the three primary studies presenting a thorough *evaluation* of the proposed approach, they are briefly discussed in the following:

- P17 - The goal of this approach is to avoid failures of a ROS-based robotic system under various scenarios. By starting from a known training set, it automatically performs inference and monitoring of specialized invariants during the lifetime of the system. The approach has been evaluated in the context of two case studies. The first case study is about a real UAV (i.e., an Ascending Technologies Hummingbird) landing on a moving platform (realized as an iRobot Create with a mounted landing platform) under different scenarios (e.g., normal, wind blowing, fragile platform, occupied platform, false airport), whereas the second case study is about a water sampling UAV, where a combination of ultrasonic, air-pressure, GPS, and conductivity sensors are used.

- P54 - This approach makes use of model-based testing and diagnosis for supporting the dependability of autonomous robots along the whole life cycle. The approach has been evaluated in the context of a real industrial installation of autonomous transport robots in a warehouse; the system includes a fleet of individual autonomous robots, a conveyor for transportation, and a central station.

- P57 - This approach presents HAZOP-UML, a method for the safety analysis of human-robot interaction; the method supports safety analysts in specifying dynamic models of the system in UML, and in identifying hazards, recommendations, and hypotheses of possible deviations of the the system from the specified dynamic

models. The approach has been evaluated by recruiting professional safety analysts and letting them apply the proposed approach on three different case studies involving (i) an assistive robot for the autonomous movement of the elderly, (ii) a KUKA Omnirob mobile robot with a KIKA Light Weight Robot arm used in workshops or factories with human workers, and (iii) a custom robot capable of navigating autonomously within a manufacturing setting while avoiding human workers, and taking and placing part boxes either on shelves or on its own base.

3.4.3.3 Validation/evaluation strategies

The analyzed studies apply different strategies for assessing their proposed approaches, independently of whether they are performed in the context of validation or evaluation research. More specifically, our analysis revealed the following assessment strategies (in order of potential realism): (i) proof of concept implementation running on simple examples, (ii) simulation-based execution and experimentation of the system, (iii) laboratory experiment where real robots are used but in a controlled environment, and (iv) realized system deployed and running in real environment.

As shown in Figure 3.10.b, the majority of the studies assess the approach in the lab (18/58), followed by proof of concept and simulation-based validations (18/58), and experiments on real deployments (4/58).

It goes without saying that validating research results in a real deployment is the best case in terms of potential for industrial adoption, and the authors of 4 studies managed to achieve this very ambitious goal (P17, P45, P54, P57). Nevertheless, we have also to acknowledge that in some cases this kind of strategy is not practical if not feasible, for example in large-scale systems involving safety issues (e.g., a fleet of flying drones in a tactical environment). These are the situations were laboratory experiments may be performed in a more manageable manner. Also, it is important to say that recently simulation environments are gaining a lot of attention thanks to the great advances they are making in terms of realism of the simulation, configurability, and possibility to run software- or hardware-in the loop simulations. The latter are enabled by the high level of decoupling provided by platforms or communication middleware like ROS, where engineers can use the same software stack as the one used in real deployments, while simulating only the components depending on the real world (e.g., drivers for the GPS, accelerometers).

The relatively high number of strategies based on proofs of concept (18/58) is somehow disappointing, specially in light of todays wide availability of software platforms, simulators, and low-cost hardware components. Assessing a scientific result via a simple

proof of concept and an example is not acceptable anymore in our research community. We expect that in future researchers on safety for mobile robots will move on from this comfort zone and will start providing more tangible (empirical) results and benchmarks about the performance of their proposed solutions. This will surely boost the potential for industrial adoption of our research.

3.4.3.4 Technology readiness level (TRL)

The purpose of the TRL is to objectively assess the maturity of a particular technology [51] on a scale ranging from 1 (minimum) to 9 (maximum). In order to keep the data extraction activity manageable and less time consuming, in the context of this work we classify the TRL of each primary study on a 3-levels scale: (i) *low* TRL (i.e., TRL $\leq$ 4), where a technology is either formulated, validated or demonstrated at most in lab, (ii) *medium* TRL (i.e., 5 $\leq$ TRL $\leq$ 6), where a technology is either validated or demonstrated in a relevant environment, and (iii) *high* TRL (i.e., TRL $\geq$ 7), where the technology is either completed, demonstrated, or proven in operational environment.

Figure 3.10.c shows the distribution of the TRL levels of our primary studies. The obtained results are self explicative, the majority of approaches (53/58) have a low readiness level, whereas only two of them are in the medium (P51, P53) and high (P17, P54, P57) levels of TRL. This is a confirmation of the results about the evaluation and validation strategies; again, if we aim at making our research products adoptable by industry, we will need to work on their technological readiness with well tested and designed tools, and realistic experimentation.

3.4.3.5 Rigor and Industrial Relevance

Rigor is defined as the precision, exactness, or correctness of use of the research method applied in a scientific work [47]. Intuitively, an experiment reported in such a way that its operational context is defined, its design is clear, and its threats to validity are explicitly discussed has higher rigor than an informal description of a running example. The main rationale for considering rigor in our research is that a primary study with high rigor is easier and more straightforward to be assessed by practitioners. Based on [47], the rigor of each primary study has been assessed according to the criteria in Table 3.9, where each criteria can be scored with the following score levels: strong (1 point), medium (0.5 point), weak (0 points). Thus, a primary study can have a rigor score ranging from 0 to 3.

Criteria	Description
Context	Is the context described to the degree where a reader can understand and compare it to another context?
Study design	Is the study design described to the degree where a reader can understand its main parts, e.g., variables, treatments, etc.?
Validity	Is the validity and threats of the study discussed and measured in details?

TABLE 3.9: Rigor assessment criteria [47]

The upper part of Figure 3.11 shows how the considered primary studies are distributed in terms of total rigor score. Here we can notice that the majority of primary studies (42/58) have a score between 0.5 and 1.5, with a mean of 1.27. Also, only 5 studies have a rigor score above 2 (P2, P31, P33, P44, P47). This result is already quite interesting: it clearly shows that researchers on safety for mobile robots should improve in terms of rigor (e.g., precision and correctness) when evaluating their research results. It also means that the majority of evaluations performed in our primary studies are either (i) experiments where rigor-related aspects are poorly reported or (ii) simple applications of the proposed approaches to toy examples. This is a clear call to researchers in the field to both better report their experiments and to focus on key aspects of the proposed approach (e.g., managed hazards, types and quality of safety-related solutions), rather than simply illustrating its application to an example.

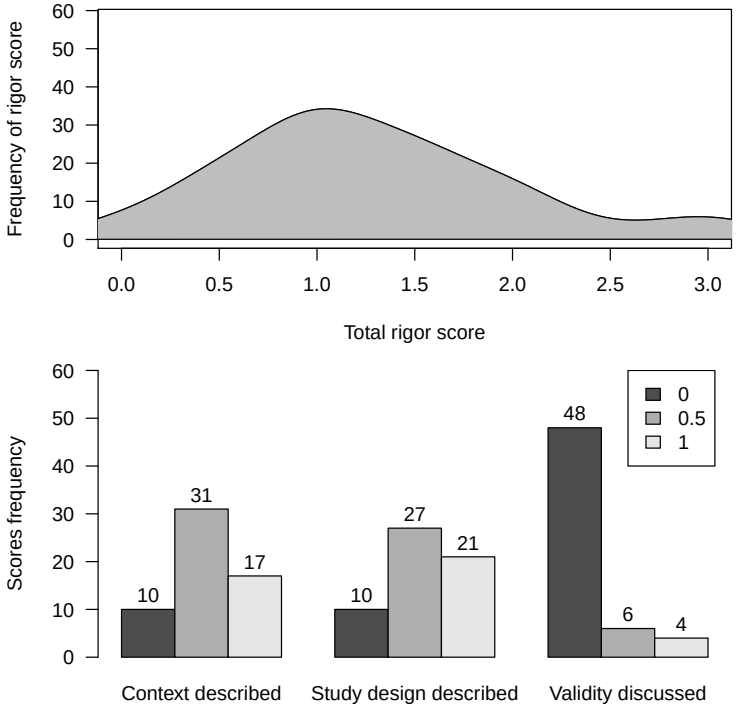


FIGURE 3.11: Results for rigor of evaluation

In order to better understand this phenomenon, we dig into the scores of all the criteria

for rigor of evaluation. As shown in the lower part of Figure 3.11, the context and the study design are performing quite well, with the majority of studies falling within the medium/strong score levels. The real problem with rigor lies in the identification and reporting of the validity of the performed evaluations; indeed during this research we seldom noticed that the threats to validity of the performed experiment have been thoroughly discussed. Of course, understanding how valid the results of an evaluation/-experiment are is a fundamental aspect for the adoptability of a proposed approach. As a solution, we suggest researchers to carefully consider all the potential threats to validity of their performed evaluations and to explicitly report them; as suggested in [16], this activity should be already carried out in the planning phase of an evaluation/experiment. Also, for easing the design, understanding and replicability of the performed evaluations, it is suggested to structure the discussion of threats to validity according to well-known classification schemes, such as the one by Cook and Campbell [52].

Industrial relevance refers to the realism of the evaluation of an approach, and determines the potential relevance of its results for industry [47]. Intuitively, an experiment involving a large number of professionals as subjects and deploying the robots in a real operational environment has a higher industrial relevance with respect to a software simulation performed in a research lab.

Criteria	**Description**
Subjects	Are subjects used in the evaluation representative (real robots)?
Context	Is the evaluation performed in a representative setting (e.g., real deployment environment)?
Scale	Is the scale of the applications used in the evaluation of realistic size (e.g., size of the operational environment, number of involved robots)?
Research method	Does the applied research method facilitate the investigation of real situations (e.g., an industrial case study)?

TABLE 3.10: Industrial relevance assessment criteria [47]

Table 3.10 shows the criteria we used for assessing the industrial relevance of each primary study. In conformance with [47], we assessed a primary study for each industrial relevance criterion as either strong (1 point) or weak (0 points). A primary study can have an industrial relevance score ranging from 0 to 4.

Similarly to the rigor score, the distribution of the primary studies with respect to their total industrial relevance score is not showing good results. Indeed, referring to the upper part of Figure 3.12, the majority of primary studies (54/58) scores lower than 2. If we zoom into the specific criteria, in the lower part of Figure 3.12 we can notice that research on safety for mobile robots suffers in terms of the context, scale, and research method dimensions. More specifically, it emerged that almost all primary studies do not

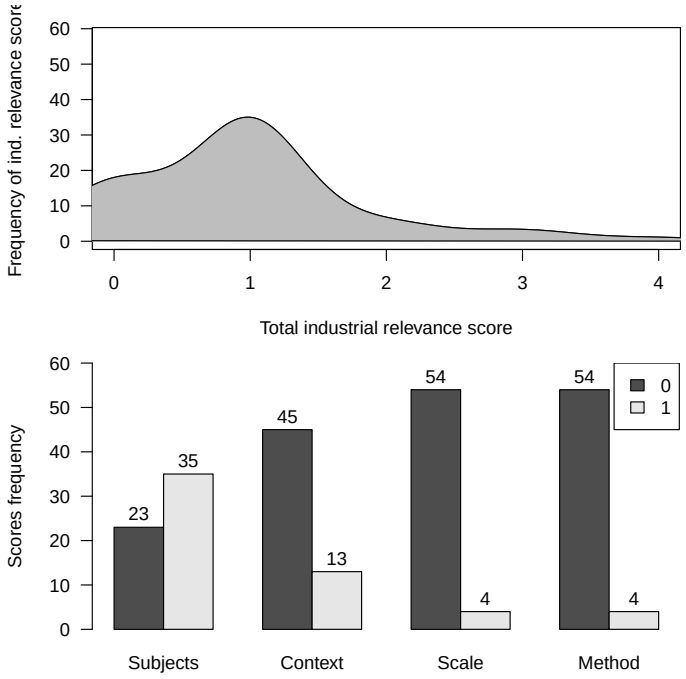


FIGURE 3.12: Results for industrial relevance

report on the evaluation of the proposed approach in a representative setting (*context* criterion, 45 studies), with a realistic size (*scale* criterion, 54 studies), or facilitating a real investigation (*research* method, 54 studies). These are all aspects that researchers should take into consideration if their aim is to develop methods that should be adopted in real industrial settings. On a positive side, the *subjects* score has a good performance. Researchers achieved this result by using in many cases real robots for their evaluations. This can be seen as a consequence of opportunities opened by open software/hardware platforms for robotics, making them accessible at low prices and with the needed level of configurability.

3.4.3.6 Industry involvement

In this section we aim at characterizing the involvement of practitioners into research studies on safety for mobile robots. Inspired by the classification used in a previous work [53], we categorize each primary study as: *academic* if all authors are affiliated with universities or research centers, *industrial* if all authors are affiliated with some companies, or a *mix* of the previous two categories.

Figure 3.13 shows that almost all primary studies contribute with an academic-only perspective (52/58). Then, only 5 studies contribute with a mixed perspective and only one study provides an industry-only perspective. This result is somehow aligned with the analysis of the previous aspects and it clearly shows the low involvement of

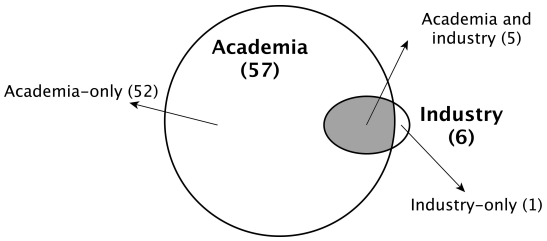


FIGURE 3.13: Distribution of industry involvement

industrial partners in research on safety for mobile robotic systems. This result is a sign of a missed opportunity; research on this research area seems to have been performed in isolation with respect to the industrial perspective, which may bring new relevant problems to be solved and a much clear picture of the state of the practice in the field. Researchers and practitioners on safety for mobile robots should work together on creating better synergies and cooperation plans so that research will be performed on industrially relevant problems and new research methods, technologies and tools will smoothly transition from academia to industry [16].

Highlights - Industrial adoption of existing approaches for safety of MRSs (RQ3): Most of the primary studies validated the proposed approaches in the lab and very few considered real deployments. Consequently, important efforts have to be spent to transfer the approaches, that currently were validated by means of proof-of-concept implementations, to real-world industrial contexts. To this end, a more significant involvement of industrial partners in the development of techniques and approaches for the management of safety for MRSs is needed.

3.5 Discussion

In this section we discuss the main findings of the paper as well as their implications for future research.

3.5.1 Single vs Multi-robots

As discussed in Section 3.4.2.12, most of the primary studies focus on a single robot (45/58). We acknowledge that there is the need of solutions to manage safety at the level of single robot, however, there is also the need of approaches that deal with multiple robots. In fact, the collaborative smart robots market size is expected to reach USD 1.07 billion by 2020 whereas the software market size for smart robots is expected to

grow at a CAGR of 30.24% from 2015 to 2020 [54]. A representative example is from Australia where a MRS was recently used for fighting the fire (http://goo.gl/EPtxgK): a five-pound Indago multicopter spotted the fire, while the unmanned K-Max helicopter dropped water on the flames.

As implication for future research, we highlight the need of solutions addressing safety when multiple robots need to collaborate with each other in order to accomplish complex missions. These approaches might require cooperative safety management mechanisms (see Section 3.4.2.2) and cooperative or centralized world knowledge (see Section 3.4.2.1).

3.5.2 Openness and capability to cope with uncertainty

In the near future, MRSs will be used in tasks of everyday life. This means that often MRSs will be used in unknown or partially unknown environments that might be shared with humans or other robots. This will require context awareness, and most of the approaches in our primary studies (41/58) have these capabilities (see Section 3.4.2.8), and adaptiveness capabilities to changing environments. As shown in Section 3.4.2.9, 25/58 approaches do not support adaptiveness capabilities and 4 approaches provide no information. Moreover, as shown in Section 3.4.2.7, only 5 approaches out of 58 are able to deal with open systems, meaning that in those cases new robots or human actors can be added at runtime.

As implication for future research, the adoption of MRSs in tasks of everyday life will require more investigation in adaptiveness capabilities as well as in dealing with open systems.

3.5.3 Compliance to standards

MRSs are very complex systems and consequently advanced techniques and tools are needed for supporting their development. Especially for critical systems, safety represents a crucial aspect to be managed since the early stages of development. In this respect, over the last decade several standards have been issued to manage MRSs safety. As shown in Table 3.7, dozens of standards are available for safety. This is justified by the fact that it is improbable to have only one standard for each quality aspect to be managed: each application domain has its own specificities that justify the need of dedicated standards. Thus, in the future we foresee the definition of further standards due to the increasing adoption of robotic systems in different application scenarios.

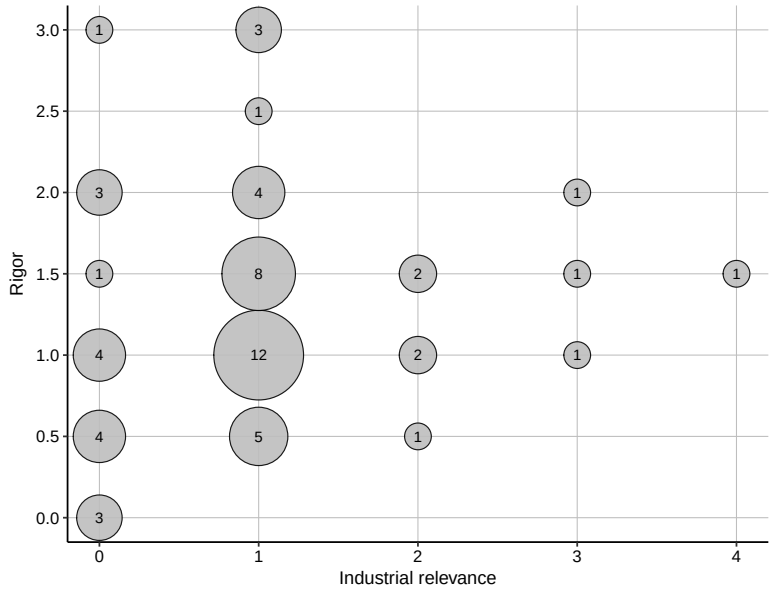


FIGURE 3.14: Aggregation of scores for rigor and industrial relevance

3.5.4 Adoption of model-driven engineering for robotic systems

Model-Driven Engineering refers to the systematic use of models as first-class entities throughout the software engineering life cycle. The objective is to increase productivity and reduce time to market by enabling the development of complex systems by means of models defined with concepts that are much less bound to the underlying implementation technology and are much closer to the problem domain. According to our study, models are currently used in the domain of robotic systems for different purposes e.g., to support the specification of missions to be executed by robots, safety constraints, etc. Most of the analysed approaches (38 out of 48) take advantage of models for performing analysis tasks since the early stages of development. This is justified by the fact that models help the understanding of complex problems and their potential solutions through abstractions. In the future we foresee the adoption of models also at run-time, as also envisioned by the Robotics 2020 – Multi-Annual Roadmap[5], e.g, to monitor and explain what robots are doing during the execution of defined missions.

3.5.5 Rigor and Industrial Relevance

As discussed in Section 3.4.3.5, the majority of evaluations in safety for robotic systems lack both rigor and relevance. This result is even more evident when considering these two dimensions together. The bubble chart in Figure 3.14 graphically shows the aggregation of rigor and industrial relevance of the primary studies. Here the majority of the

[5]https://www.eu-robotics.net/cms/upload/downloads/ppp-documents/Multi-Annual_Roadmap2020_ICT-24_Rev_B_full.pdf

primary studies falls in the lower-left quadrant, highlighting the lack of both rigor and industrial relevance. Given the situation, in the following we propose a set of strategies for improving the evaluation of robotic systems in terms of both rigor (i.e., moving ↑ in Figure 3.14) and industrial relevance (i.e., moving → in Figure 3.14):

- improve the design of the performed experiments by, e.g., formalizing the safety hazards being considered, explicitly defining the dependent/independent variables of their experiments, identifying sound statistical analyses of the obtained data (↑);
- elaborate on and discuss potential threats to validity before and after evaluating the robotic system (↑);
- improve the measurement precision when performing experiments involving both the software and hardware parts of the robots (↑);
- carefully select the software and hardware platforms for the evaluation, preferably using real robots (→);
- carefully select realistic operational environments where the robots will be deployed (→);
- push towards large-scale, or at least realistic-scale evaluations, involving a realistic number of robots and involved human users, this is especially true for swarm and multi-robot systems (→);
- when possible, push towards investigating real situations involving industrial partners, practitioners, and in-the-field operators (→).

Researchers can use the above mentioned strategies to ensure an adequate rigor and relevance when planning the evaluations of approaches for safety of robotic systems.

3.5.6 Software engineering and robotics

As stated by the H2020 Multi-Annual Robotics Roadmap ICT-2016 [55], in the production of software for robotic systems "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 use of ad-hoc development processes in general, and software engineering approaches in particular, hampers reuse and complicates the configurability of existing solutions. This justifies the need of systematic approaches, methods, and

tools to (i) easily configure robots, or provide them with self-configuration capabilities, (ii) specify robotic tasks in an easy and user-friendly way, and (iii) make the robots able to take decisions on their own to manage unpredictable situations. This shifts towards well-defined engineering approaches will stimulate component supply-chains and significantly impact the robotics marketplace.

Even though there is a growing interest (see Section 3.4.1), the community of software engineering and robotic is still not consolidated. This is testified by the extreme fragmentation of the targeted publication venues, as discussed in Section 3.4.1. There are some workshops and initiatives in the direction of creating a community around software engineering ad robotics, such as the international workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob), the series of workshops on Model-Driven Robot Software Engineering (MORSE), the Journal of Software Engineering for Robotics (Joser), and a recent technical briefing at ICSE on software engineering for robotic systems [56]. However, more work is needed in order to create a proper community on this topic.

3.6 Threats to Validity

The quality of our research has been ensured by defining a complete research protocol beforehand, by letting it assess by independent reviewers, and by conducting research following well-accepted guidelines of systematic review/mapping study [16, 34, 35]. Also, to allow independent replication and verification of our study, a complete replication package is publicly available[6] to interested researchers. Our replication package includes the review protocol, the list of all considered and selected studies, the description of the parameters for the data extraction activity (i.e., the data extraction form), the raw extracted data, and the R scripts for data analysis.

In the following we discuss how we considered and mitigated the potential threats to validity of our study by following the Cook and Campbell classification framework for threats to validity [16].

Conclusion validity. Conclusion validity refers to the relationship between the extracted and synthesized data and the produced map and findings [16].

In order to mitigate possible conclusion biases, first of all we sistematically defined the search string of our automatic search (see Section 3.3.2) and we documented all the steps of our research in a publicly available research protocol. This allows third-party researchers to replicate our study independently.

[6] http://cs.gssi.it/safetyMRSReplicationPackage

Moreover, we documented and we used a rigorously defined data extraction form, so that we have been able to reduce possible biases that may happen during the data extraction process; also, in so doing the data extraction process can be considered as consistent and relevant to our research questions.

On the same line, the classification framework may be another source of threats to the conclusion validity of our study; indeed, other researchers may identify classification frameworks with different facets and attributes. In this context, we are mitigating this bias by (i) performing an external evaluation by independent researchers who are not directly involved in our research (see Section 3.3, and (ii) having the data extraction process conducted by the principle researcher and validated by the secondary researcher.

Internal validity. Internal validity is concerned with the degree of control of our study design with respect to potential extraneous variables influencing the study itself.

In this case, having a rigorously defined protocol with a rigorous data extraction form helped in mitigating biases related to the internal validity of our research. Also, for what concerns the data analysis validity, the threats are minimal since we employed only descriptive statistics when dealing with quantitative data. When considering qualitative data, we performed a thorough sensitivity analysis on all extracted data in order to have a coherent set of information both for each primary study and across studies.

Construct validity. Construct validity concerns the validity of extracted and synthesized data with respect to our research questions. Construct validity concerns the selection of the primary studies with respect to how they really represent the population in light of what is investigated according to the research questions.

Firstly, we are reasonably confident about the construction of the search string used in our automatic search since the used terms (e.g., safety, mobile robotic system, etc.) have been piloted in preliminary searches (using the IEEE Xplore library); also, the chosen terms of the search string have been evaluated by the reviewers of our research protocol beforehand. As described in Section 3.3.2, the automatic search has been performed on multiple electronic databases to get relevant studies independently of publishers' policies and business concerns. The used electronic databases cover the area of software engineering well [40, 41], and we are reasonably confident that this applies also to safety for mobile robotic systems from the software engineering point of view.

Moreover, we complemented the automatic search with the snowballing activity performed in stage 3 of our study search and selection process (see Figure 3.2), thus making us even more confident about the search strategy of this study. Since our automated search strategy actually relies on the quality of the used search engines and on how

researchers write their abstracts, the set of primary selected studies has been extended by means of the multi-step snowballing procedure (see stage 2 in Figure 3.2).

After having collected all relevant studies from the automatic search, we rigorously screened them according to well-documented inclusion and exclusion criteria (see Section 3.3.2); this selection stage has been performed by the principle researcher, under the supervision of the secondary researcher. Also, in order to assess the quality of the selection process, both principle and secondary researchers assessed a random sample of studies, and the inter-researcher agreement has be statistically measured with good results (see Section 3.3.2).

Because of all the above mentioned strategies for mitigating possible threats to the construct validity of our research, we are reasonably confident that we unlikely missed potentially relevant studies.

External validity. It concerns the generizability of the produced map and of the discovered findings [16].

In our research, the most severe threat related to external validity consists in having a set of primary studies that is not representative of the whole research on safety for mobile robotic systems. In order to mitigate this possible threat, we employed a search strategy consisting of both automatic search and double-step snowballing of the primary selected studies. Also, having a set of well-defined inclusion and exclusion criteria contributed to the external validity of our study.

Moreover, only studies published in the English language have be selected in our search process. This decision may result in a possible threat to validity because potentially important primary studies published in other languages have not been selected in our research. However, the English language is the most widely used language for scientific papers, so this bias can be reasonably considered as minimal.

Similarly, grey literature (e.g., white papers, not-peer-reviewed scientific publications) is not included in our research; this potential bias is intrinsic to our study design, since we want to focus exclusively on the state of the art presented in high-quality scientific papers, and thus undergoing a rigorous peer-reviewed publication process is an accepted requirement for this kind of scientific works.

3.7 Related Work

In this section we discuss those secondary studies which completely or partially are addressing the topic of safety in MRSs.

The authors of [57] present a general survey of various publications that focus on mechanical design and actuation, controller design and safety criteria and metrics used to validate safety of a domestic robot during unexpected collisions between a robot and a human user, without elaborating the separate papers in details. Furthermore, the focus on the survey is on the mechanical and controller design, while not taking in consideration safety from a software engineering point of view.

A review about Human-Robot Interaction (HRI) is presented in [58]. It attempts to identify the key themes and challenges from multiple perspectives, as HRI requires understanding and comprehension of multiple domains related to people, robotics, design, cognitive psychology etc. It states that although the review follows survey structure there are necessarily well-written and influential papers that are not referenced and covered.

Also, a survey investigating safety issues in human-robot interactions is proposed in [59]. It starts with a review of safety issues in industrial settings, then shifting focus on safety issues related to mobile robots that operate in dynamic and unpredictable environments. It gives general ideas and directions of possible hazards and methods used for risk reduction, pointing out risks being introduced with the development of modern robotic systems.

Finally, the authors of [60] present the state of the art and enlighten a number of challenges in the field of safe and dependable physical human-robot interaction undertaken within two projects: PHRIDOM (Physical Human-Robot Interaction in Anthropic Domains) and PHRIENDS (Physical Human-Robot Interaction: dependability and safety). Results from different research groups about possible metrics for the evaluation of safety, dependability and performance in physical human-robot interaction are presented. The sources for the discussion on physical human-robot interaction is based on number of articles taken from predetermined workshops, European projects and journals.

3.8 Conclusions

In the near future, MRSs will need to be able to operate in uncontrollable and unknown environments. Moreover, often MRSs will be required to collaborate both with each other and with humans, to accomplish complex missions. In the last decades, robotic research has made huge progresses. However, as this study testifies, existing solutions are not yet ready to be used in everyday life, and in uncontrollable and unknown environments often shared with humans. We came to this conclusion through a mapping

study devoted at investigating how existing solutions for MRSs address safety aspects. Specifically, the three research questions we investigated are:

- **RQ1**: *What are publication and research trends of the studies about safety for mobile robotic systems?*
- **RQ2**: *How do existing approaches address safety for MRSs?*
- **RQ3**: *What is the potential for industrial adoption of existing approaches for safety for MRSs?*

The results provided by answering RQ1 give a sharp and precise overview about the scientific interest and research trends in the area of safety for MRSs. The classification resulting from our investigation on RQ2 provides a solid foundation for *researchers* willing to further contribute this research area with new approaches for safety for MRSs, or willing to better understand or refine existing ones. Our results with respect to RQ3 can be of special interest for *practitioners* since together they provide an evidence-based instrument for identifying which approaches for safety for MRSs are more ready to be transferred to industry.

The main findings and implications of this study are:

- (**R1-I1**) We found that most of the approaches surveyed in this study focus on a single robot. Therefore, when multiple robots need to collaborate each other in order to accomplish complex missions, it emerges then the need of solutions addressing safety for MRSs.
- (**R2-I2**) Many of the surveyed approaches do not support adaptiveness capabilities and most of them are not able to deal with systems supporting the addition and removal of robots, human actors, etc. at runtime. Tasks of everyday life will require more investigation in safety-oriented adaptiveness capabilities of MRSs.
- (**R3-I3**) Many domain-specific standards related to safety are currently available. However, only a minority of the surveyed approaches are compliant to standards targeting safety aspects. Consequently, when developing a robotic system, specific standards have to be taken into account to make it compliant to them and safe for the considered application domain.
- (**R4-I4**) The majority of evaluations in safety for robotic systems lack both rigor and relevance. Therefore, there is the need of new strategies to better support and planning the evaluations of approaches for safety of robotic systems.

- (**R5-I5**) Even though there is a growing interest and some relevant initiatives, the community of Software Engineering for robotics is still not consolidated. The challenge for the research community is to promote a shift towards well-defined engineering approaches able to stimulate component supply-chains and significantly impact the robotics marketplace.

In summary, this study provides a comprehensive and replicable picture of the state of the art on safety for MRSs, helping researchers and practitioners in finding characteristics, limitations, and gaps of current research on safety for MRSs. We believe and we hope that the results of this study will lead to the development of new methods and techniques for safety for MRSs, making them one step closer to supporting us in our everyday tasks of the near future.

Chapter 4

Design-time Modeling and Analysis

MMRSs are increasingly popular since they promise to simplify a myriad of everyday tasks. At the state of the art and practice on-site operators must deeply know all the types of used robot in terms of e.g., flight dynamics and hardware capabilities in order to correctly operate with them. On-site operators have to simultaneously control a large number of robots during the mission execution. Nowadays, vendors provide low-level APIs and basic primitives to program robots, thus making mission development an (error-prone) activity that can be performed by a restricted number of highly skilled professional stakeholders. As clearly stated in the Robotics 2020 - Multi-Annual Roadmap For Robotics in Europe[1]:" *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"*. Furthermore, tasks are very specific and this limits the possibilities for their reuse across missions and organizations. Consequently, current approaches are affordable only for users that have a strong expertise in the dynamics and technical characteristics of the used robots. Software engineering approaches and methodologies are needed to support the definition, the development, and the realization of missions involving teams of autonomous robots while guaranteeing safety.

In this chapter we describe the first part of our modeling framework that supports the *specification of missions* in MMRSs. In parallel with the framework a tool at University of L'Aquila is developed (FLYAQ) that enables a definition of missions for MMRSs and generation of detailed flight plan for each robot in the system. Specifically, starting from a high-level description of the mission, FLYAQ automatically generates a detailed

[1]http://sparc-robotics.eu/roadmap/

flight plan for a team of autonomous robots that will satisfy the specified mission while preventing collisions with other robots and obstacles, and respecting no-pass zones. The development of the tool is based on a specific type of autonomous robots - multicopters.
.

This work builds on top of the FLYAQ platform [3, 61][2] that makes possible the specification of missions for end-users with expertise neither in ICT nor in robots dynamics, e.g., fire-fighters and rescue workers. FLYAQ ensures a strong adherence with the application domain by providing an extensible domain specific language, named Monitoring Mission Language (MML), which permits to graphically define the missions. Extension mechanisms of the language allow domain experts to specialize MML with additional tasks that are specifically tailored to the considered domain. For example, if operators are interested on monitoring solar panel installations in a rural environment, the language might be extended with tasks representing the concept of, e.g., solar panel groups, thermal image acquisition, solar panel damage discovery and notification. End-users of FLYAQ do not need to have expertise either in ICT or in robots' dynamics.

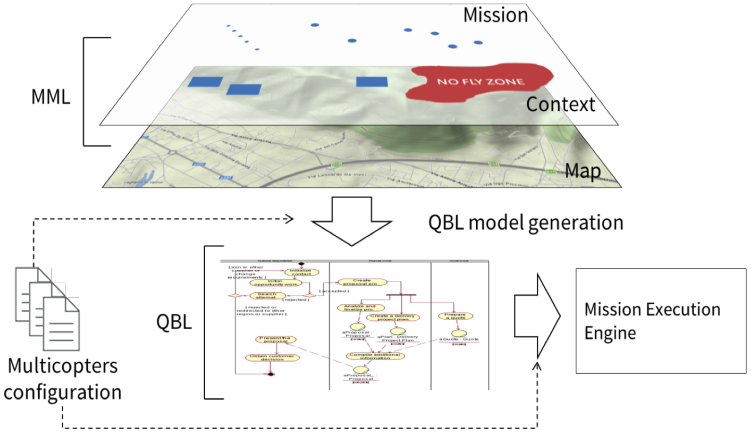


FIGURE 4.1: Overview of the FLYAQ

As shown in Figure 4.1, MML is composed of three layers:

- to specify the mission by means of the modeling constructs provided by the language
- to specify the context in which MMRSs have to operate such as no-pass zones, obstacles, etc.
- a map representing the geographical zone where the mission will be executed.

[2] http://www.flyaq.it

Once the mission has been graphically specified, then waypoints and trajectories have to be calculated. They are represented in an intermediate language called Behaviour Language (BL). Examples of BL actions include: start, stop, goto, read from a sensor, send/receive a notification to/from other robots, etc. BL has been defined through an iterative process involving experts of the domain within the FLYAQ project which is focused on mission planning of autonomous multicopters. The BL model is the input to a set of software controllers, each of them commanding a single robot, depending on the various movements and actions contained into the BL model. Each controller is dedicated to a specific type of robot so that it is able to account for the specific dynamics and other characteristics of the managed robot. This aspect of the system is not in the scope of this work, so in the following we will not focus on the hardware and low-level features of the robots involved in our missions.

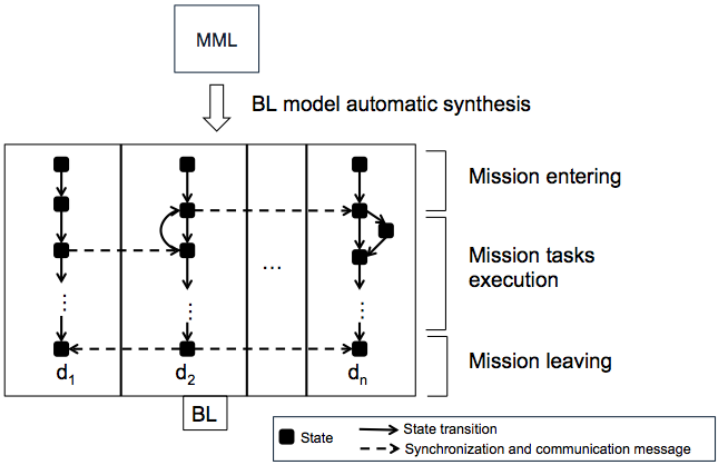


FIGURE 4.2: BL model automatic generation

As shown in Figure 4.2, the BL model automatically generated out of the MML model is organized into n parts, each of them describing the detailed plan of a specific robot. The plan of a robot can be abstracted as a finite state transition system where each transition corresponds to a BL operation. Both cyclic and alternative behaviour operations can be performed. Robots can exchange synchronization and communication messages (see dashed arrows in the Figure 4.2). FLYAQ assumes that at the beginning of the mission each robot will be positioned at its home location and that at the end of the mission it will stop to a (possibly different) location. Then, the finite state system is structured in three parts:

- **Mission entering** consisting of the operations required to start the mission, e.g., take-off;

- **Mission tasks execution** consisting of the operations required to accomplish each mission-specific task, e.g., searching for an object in an area, taking a picture in the waypoints of an area, detecting the level of carbon dioxide in a specific point;
- **Mission exiting** consisting of the operations required to conclude the mission, e.g., going back to home location, landing etc.

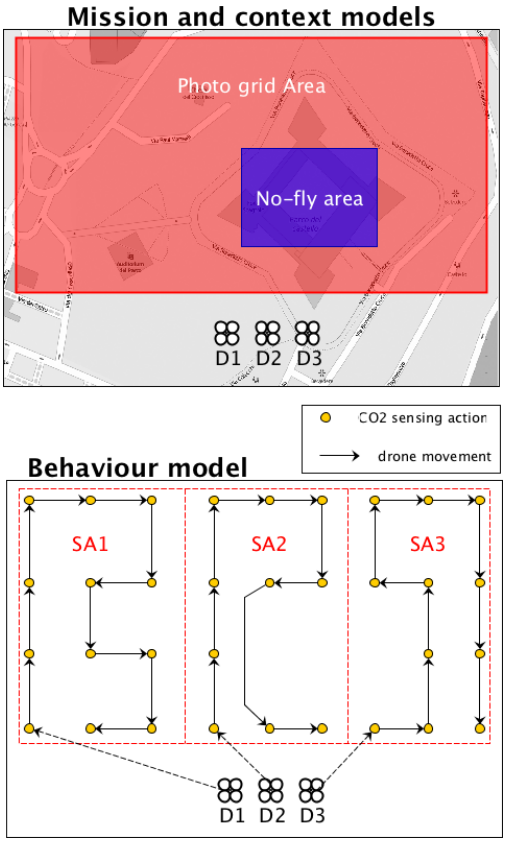


FIGURE 4.3: Behaviour Generation

The main difficulty of the behaviour generation process Figure 4.3 is that MML is extensible and hence it is not possible to define once all the translation rules needed to translate MML tasks into BL operations. Therefore, the automated generation is based on three main concepts:

- typology and characteristics of the region that is affected by the mission
- strategies to be applied to calculate the concrete movements that robots have to perform
- actions to be performed while visiting the identified waypoints.

These concepts pose constraints and permit to use consolidated and optimised algorithms (e.g., shortest-path calculation between two points, path planning, etc.) that will be exploited by the generation process to, e.g., determine how a robot should visit a region according to specific path planning policies. A platform extender can define concrete tasks for the considered domain by associating these tasks to the general concepts defined in the platform. In this way, as described in details in the next section, it is possible to completely automate the generation of a *safe robot behaviour specification.*

4.1 Design Elements: Terminology & Definitions

Let $Coordinates = \{(x, y, z)|x, y, z \in R\}$ be the universal set of geographical coordinates where the mission takes place; for $C \in Coordinates$, $c.x$ denotes the longitude of c, $c.y$ is the latitude, and $c.z$ is the altitude. To represent the mission and the context in which a MMRS performs the mission we define a region as a generic term for defining geometric shapes in the environment.

Definition 4.1 (Region)**.** A region $R = (P, t)$ is a pair where $P \in Coordinates$ denotes a geometric shape in the environment (ex. it can identify a polygon in the 2D environment or a polyhedron in the 3D one) and t defines the type of the region (ex. obstacle, no-pass zone, mission region etc).

In our work, we use regions to represent the geographical area of the mission and its context, more specifically to represent three types of logical structures in the definition of the mission: (i) an *obstacle*, (ii) a *no-pass zone*, (ii) *mission region.* We define an *obstacle* as a region in the environment that should not intersect with the region of operation of an active agent(robot) because has negative impact on safety. *No-pass* zone is a region in the environment that should not intersect with the region of operation of an active agent(robot) because of law regulations, but does not have impact on safety. *Mission region* is the minimum bounding box of the areas of all tasks.

Depending on the mission application scenario, different types of geometries are supported by MML: point, line, polygon, volume to represent regions (Figure 4.4).

Now, we formally define the context in which the mission takes place.

Definition 4.2 (Context)**.** A context $C = (Nz, O)$ is a pair where $Nz \in R^n (n > 0)$ is a list of no-pass zones, and $O \in R^m (m > 0)$ is a list of regions representing obstacles.

In the following we focus on the concept of an agent, see Definition 4.4. Examples of agents in MMRSs could be entities that take participation in the mission like robots,

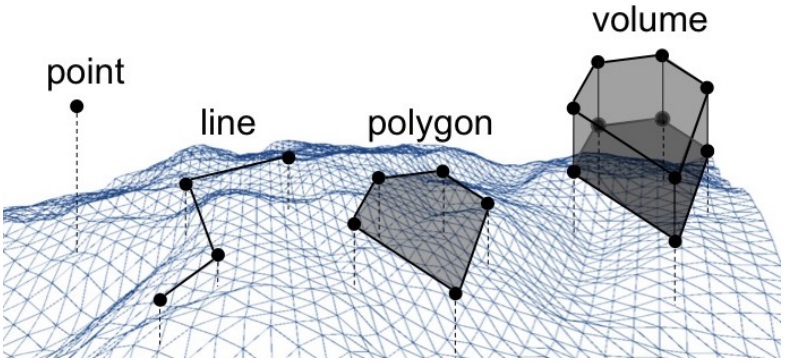


FIGURE 4.4: Geometries supported by MML

ground stations, humans, etc. However, in the design-time modeling we are focusing only on robots as entities that execute the mission. Informally, we represent an agent through its state (a set of properties representing its configuration and environment where it operates). Each property describes a particular aspect of the agent's state (e.g., an agent's location, current speed, current battery level, set of resources or information of the environment). We define property for an agent as follows.

Definition 4.3 (Property)**.** A generic property $P_i = (ID, V)$ is defined as a tuple where:

- ID is a unique identifier of the property
- V is the value the property receives. The values type depends on the nature of the property.

Now, we formally define an agent.

Definition 4.4 (Agent)**.** A generic agent $a_i = (CP, EP)$ is defined as a tuple where:

- CP is the agent's configuration expressed as a set of uniquely identified *configuration properties* (ex. current speed, current battery level etc.)
- EP is a set of uniquely identified *environmental properties* each describing a particular aspect of agent's environment. An agent observes and collects information about its local environment (ex. obstacles, other agents in the local environment etc.).

In the following we focus on the concept of task, see Definition 4.5. Informally, a concrete Task specifies (i) the region defined as a *goal space* in which the involved agents(robots) have to perform actions (ii) the strategy according to which the region will be be decomposed in sub-regions (iii) the action performed by the agents(robots).

Let STR be the universal set of strategies for region decomposition, and let ACT be the universal set of actions that realize concrete robot actions (like taking a picture with a given resolution, detecting the presence of carbon dioxide, etc.). $\perp \in ACT$ denotes null action meaning that it is ineffectual. A task is defined as follows.

Definition 4.5 (Task)**.** A task $t = (G, S, act)$ is a tuple where:

- G is the region named as a *goal space* in which the involved agents(robots) have to perform actions
- $S \in STR$ is the strategy according to which the goal space will be be decomposed on (sub)regions
- $act \in ACT \times \{i, c\}$ specifies the concrete action to be performed by the agents(robots) involved in the task. $\{i, c\}$ identifies if the action is instantaneous, like taking a picture, or continuous, like taking a video. An instantaneous action is executed only at one point of time during mission execution. A continuous action is started at the beginning of the task and terminated at the end. In case of $\perp$, the continuous/instantaneous flag does not matter.

It is important to note that the concrete action *act* of a task t is generic, and it may correspond to a set of more specific atomic actions to be performed by the robot. For instance, an action aimed at collecting some data can correspond to two "atomic" operations at the controller level to be performed in sequence, e.g., an operation for taking a picture using the on-board camera and an operation for sensing the CO2 level from a CO2 hardware sensor.

Let $SR = \{SR_1^t, \ldots, SR_n^t\}$ is a set of subregions on which the goal space G of a task t is decomposed according to a strategy S. Hence, we have $\{SR_1^t \cup \cdots \cup SR_n^t\} = G$ and $\{SR_1^t \cap \cdots \cap SR_n^t\} = \emptyset$ meaning that the union of all subregions correspond to the goal space G, and there isn't an overlap between different (sub)regions when the goal space is decomposed.

In the following we define a **Mission**. Let *Tasks* denotes the universal set of all tasks (definition 4.5) and let A is the set of all agents (definition 4.4).

Definition 4.6 (Mission)**.** A mission $M = \{t_1, ..., t_n \mid t_1, ..., t_n \in Task\}$ is defined as a set of tasks that need to be executed.

In this work we consider that the priority of all tasks in a mission is the same. Hence, we assume that there is not any ordering of tasks and each task can be performed at the same time as any other.

Now, we formally define a MMR system. Agent is the basic unit around which a MMRS is defined.

Definition 4.7 (MMRS)**.** A **MMRS** $RS = \{(a_1, \{t_1, ..., t_k\}), ..., (a_n, \{t_1, ..., t_n\}) | a_1, ..., a_n \in A \wedge t_1, ...t_k, ..., t_n \in TASK\}$ is a set of active agents participating in mission's tasks.

Now, we formally define agent's behaviour for a task t. We denote by S^t the set of all possible states for an agent a_i performing a task t, while with OP_i^t the set of all atomic operations an agent can perform. The following definition describes the behaviour state space for an agent a_i to perform a task t with a starting state S_0.

Definition 4.8 (Behaviour state space)**.** A **Behaviour state space** of an agent a_i is a tuple $B_i^t = (S^t, \Sigma, \Delta, S_0)$ where:

- S^t is a finite, non-empty set of agents' states ;
- Σ is a finite, non-empty set of transitions;
- Δ is the transition function, and $(S_p, \lambda, S_k, I) \in \Delta$ is a transition that define the key characteristics of a Behaviour execution where:
  - S_p is the source state of the transition
  - S_k is the the target state of the transition
  - $\lambda \in \Sigma$ is a concrete atomic transition that is performed;
  - I is a set of conditions that must be satisfied for a transition to be performed. The conditions are agent's properties that capture the key characteristics of the agent state or its environment. (ex. $batterylevel > x$ etc.);
- $S_0 \in S$ is an initial state;

When an agent a_i performs an operation $op \in OP_i^t$, we can see the impact in two aspects: (i) in the physical location l_i where the action takes place and (ii) in the the agent's configuration C_i. $l_i \in EP$ is one of the environmental properties of an agent a_i (Definiton 4.4), while $C_i \in CP$ is the agent's configuration represented through one or a few properties as described in (Definiton 4.4). That is why, we represent OP_i^t=(Γ_i^t, Ω_i^t) as a tuple of two disjoint subsets where:

- Γ_i^t is the set of all "atomic" movements available for an agent a_i performing task t
- Ω_i^t is the set of all "atomic" actions available for an agent a_i performing task t

An agent performs a behaviour plan BP_i from a starting location l_i to a goal location l_j i.e. moves from its current location to a goal location by executing a sequence of "atomic" movements and "atomic" actions. In this work, a combination of an "atomic" movement and "atomic" action performed at a same time is considered as an "atomic" operation. Ex. a robot can *move* from a starting location to a goal location and *record* a video. Here, *moving* is an "atomic" movement, while *recording* a video is an "atomic" action. We consider that the combination of both of them happening at a same time is an "atomic" operation that cannot be split on smaller parts.

We explore further the transition function in Definition 4.8. For an agent state $S_i \in S^t$ and an operation $op \in OP_i^t$, we denote by $tran(S_i, op)$ the state where the agent transits when successfully performed an operation op at state S_i. We use $interTrans(S_i, op)$ to denote the set of intermediate states through which the agent traverse when performing op at a state S_i (including S_i and $tran(S_i, op)$). Now we formally define a behaviour plan.

Definition 4.9 (Behaviour Plan)**.** A behaviour plan is defined as a sequence of atomic operations to be applied to an agent A_m to transit from its current state S_k to a goal state S_j. A behaviour plan is denoted by $BP_i^t = (op_1, op_2...op_p)$, where $op_n \in OP_i^t$ for $n \in \{1, 2...p\}$

Each behaviour plan BP_i associated with a particular task t we modeled it as a separate behavioural *unit*. For each task, a separate behaviour plan for each agent should be generated as part of the mission.

4.2 Automatic generation of behaviour plans

In this section we focus on the explicit construction of the Behaviour Plan BP_i for a generic agent a_i. At the beginning, the initial and goal state S_0 and S_f are known and BP_i is empty. The Behaviour Plan BP_i is synthesized as the disjoint union of three different sets of operations:
Mission Entering (ME): Δ_e represents the BL operations to be executed for letting an agent(robot) a_i start the mission;
Mission Tasks Execution (MTE): Δ_t represents the BL operations to be executed by a_i to accomplish its assigned tasks. For each task in which a_i is involved, a_i (i) will correctly approach the starting point of the task without collisions with other agents and obstacles and without traversing no-pass zones, and (ii) it will perform operations to accomplish it;
Mission Leaving (ML): Δ_l represents the BL operations to be executed for leaving

the mission, i.e., correctly exiting from the final task by letting agent a_i come back to its home location.

At the beginning of the mission, the position of agent a_i is identified by $home_i$. The above synthesis make use of three auxiliary functions that implement suitable operations to (i) distribute the goal space G of each mission into a set of (sub-)regions, each of them assigned to a specific agent (see function **Divide** below); (ii) let an agent approach the mission by reaching the starting point in the subregion assigned to it (see function **Appr** below), and (iii) cover the assigned region according to the specified strategy and by performing the specified actions for each way point (see function **Cover** below). The generation of auxiliary functions builds on state-of-the-art and well-established algorithms for solving problems like polygon partitioning, path finding, and graph traversals [62]. The modularity of the framework allows a straightforward inclusion of alternative algorithms and/or future advances of existing ones without affecting the generation process. Let $n > 1$ be the number of agents involved in the mission, the three functions are defined as follows.

Divide: It takes as input the specification of a Task, the current positions of all the involved agents, and the specified context. Divide gives as output a tuple of subregions representing a spatial partition of the task space. The $i-th$ area is the one assigned to agent a_i. We assume that obstacles and no-pass areas can overlap with the geographical region of a task but, for the sake of simplicity, we assume that there is no no-pass zone crossing the geographical region and cutting it into two isolated subregions. Also, to keep the *Divide* function computationally lightweight, we assign each sub-region to the agent with the smallest distance to the centroid of the sub-area in the Euclidean plane.
Appr: It takes as input the current position of an agent, the task sub-region assigned to it (as retrieved by Divide), and the context. Appr gives as output the path that the agent has to perform for correctly reaching, from its original position (home location), the mission starting point in the sub-region assigned to it. This function generates the obstacle- and collision-free path that an agent a_i must travel to reach the geographical subregion assign to it. Independently from the shape of geographical area (i.e., point, line, polygon, or volume), the Appr function can be formulated as a path planning problem for multiple vehicles in the three-dimensional world [62]. Path planning is still one of the open problems in the field of autonomous systems, especially as the number of degrees of freedom increases (e.g., depending on the differential constraints depending on the vehicle movements dynamics, on its minimum and maximum speed, on the presence of other vehicles moving within the environment, etc.). Tens of path planning algorithms exist, each of them applying different kinds of heuristics; an overview and deep discussion of existing path planning algorithms can be found in [63]. Since our framework is independent of the algorithm internally used by the Appr function, for

the sake of simplicity in this work we apply a basic algorithm (leaving room for further refinements). Specifically, the algorithm works according to the following steps:

- **Obstacles enlargement:** replace each obstacle ob in the environment by its three-dimensional Minkowski sum with a sphere with radius r. The value of r is computed as being the sum of (i) the length of the maximum side of the agent a_i and (ii) an arbitrary value σ representing a safety boundary with respect to the minimum distance that a_i can fly with respect to any obstacle boundary. This operation enables us (i) to automatically discard all passages which are too narrow for the agent and (ii) to have a minimum safety distance between the agent and each obstacle.

- **Target points identification:** target points are those points that the agent can travel to reach the geographical region R. Target points are identified by firstly defining a set of potential target points TP, depending on the geometry of R (they will be considered in the last step of the basic algorithm). If R is a point, then TP contains only a itself; if a is a line, then TP contains both the first and last points of the polyline representing R; if R is a polygon, then TP contains every vertex of each open edge of the area R, as they have been identified by the Divide function; if R is a volume, firstly we consider the two polygons representing its base at the lowest and highest heights, secondly TP is computed by applying the same procedure for the case of polygons for both of them.

- **Trajectory definition:** Firstly, we compute the set SP containing the shortest paths between the current position of the agent a_i and all the target points in its TP set by iteratively applying a 3D extension of the well-known visibility graph algorithm [64]. We use the visibility graph algorithm since (i) it is optimal in terms of the length of the solution path, and (ii) its implementation is quite simple with respect to other path planning algorithms [26]. A classical caveat of the visibility graph algorithm is that it tends to take the robot as close as possible to obstacles, however we alleviate it by means of our obstacles enlargement preliminary step. Secondly, we select the shortest path among the paths in SP.

It is important to note that the trajectories generated by the Appr function should not intersect. In this context, for each calculated trajectory of an agent we consider it as an obstacle in the next iteration, that way avoiding trajectory intersection between agents, so that the agents travels each trajectory intersection at different coordinates, thus avoiding potential collisions by design. We make the assumption that there will always be a path from a robot to its assigned region that will not intersect with another robot's path. This assumption seams reasonable because in our work we focus on open

outdoor environments where there are plenty of paths between a robot and its assigned region. We are aware that this solution may also lead to potentially inefficient flight plans. However, we will release this assumption in the next section during the adaptation process making our solution more appropriate for environments where MMRSs consists of lot of robots and there are lot of obstacles in the environment. In this context, we will provide a refined version of the *Appr function* that takes into consideration also the concept of time when planning the trajectories of the agents within the environment. **Cover:** This function takes as input a starting position s of an agent a_i, a geographical region R, and a real number r representing the resolution of the grid that implicitly discretizes R. It produces a set of coordinates named goals locations to be travelled by a_i for covering the whole region R. The Cover function can be formulated as a coverage path planning problem for robotic systems. In the literature there is a large number of algorithms for solving this problem, each of them with specific benefits, drawbacks, and application domains (e.g., underwater robots, flying robots with strict flight dynamics, demining robots, etc.) [65, 66]. So, depending on the type of geometry of R, we can distinguish between the following cases: if R is a point, then the function returns R itself; if R is a line of length l, then the function return l/d points, each of them having d distance by the others. If R is a polygon, then according to the classification provided in [67], our Cover function can be realized via an off-line, optimal coverage algorithm with support for polygonal and non-rectilinear boundaries. In light of this, we identified the algorithm presented in [66] as a good candidate for our needs. Basically, it is based on the Boustrophedon cellular decomposition that ensures a complete coverage of the available free space, while minimizing the path of the robot within a known area; it also accounts for a fixed entry point of the robot, that will correspond to the starting point s. In order to cover the whole area at the right resolution, we fix the step size (i.e., the distance between two parallel line segments) for the Boustrophedon motion to the resolution r. The Cover function produces a point element along the identified portion of the whole motion plan at every d meters. Finally, if R is a volume, we firstly divide it horizontally into h/d planes, where h is the height of the volume, and then we iteratively apply the previously mentioned motion planning algorithm [66] for each identified plane. Also in this case we are applying a trade-off decisions between the complexity of the planning algorithm and its efficiency; we believe that this decision helps in keeping the proposed function easy to understand, without any dependency to complex 3D, full-space, planning algorithms, which are very few, even in the most recent works [66].

The generation from MML to BL has been implemented by following the model-driven engineering paradigm. More precisely, the outputs of the three auxiliary functions applied to the source MML model are represented as three corresponding models. Such

models are taken as input by the model transformation MM2BL, which is able to generate BL models out of MML ones. Such transformation is developed by means of the Atlas Transformation Language (ATL) [68], which is a hybrid language containing declarative and imperative constructs. The fragment of the MM2BL transformation consists of a header section, transformation rules, and a number of helpers, which are used to navigate models and to define complex calculations on them.

4.3 Evaluation of generated models

As a first step towards a realistic assessment of the feasibility of our automatic generation method, we specified missions for autonomous multicopters in FLYAQ (we used multicopters as a type of agents(robots) for our evaluation). We modelled missions with MML and executed generated BL models by using a Software-InThe-Loop (SITL) simulation platform. The main characteristic of SITL simulations is that the used software stack is exactly the same as the one used in real flights; the only difference with respect to real flights is that the key low level hardware drivers (e.g., GPS sensors, accelerometers, etc.) are simulated via a dedicated software. Figure 4.5 shows the modules of our SITL simulation setup.

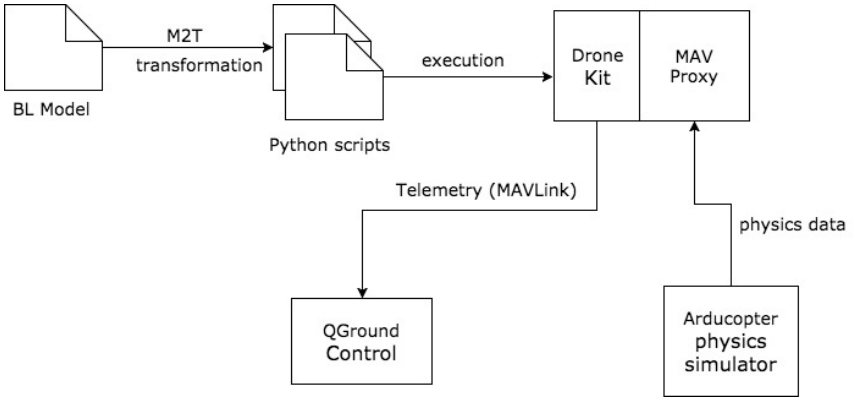


FIGURE 4.5: SITL simulation stack

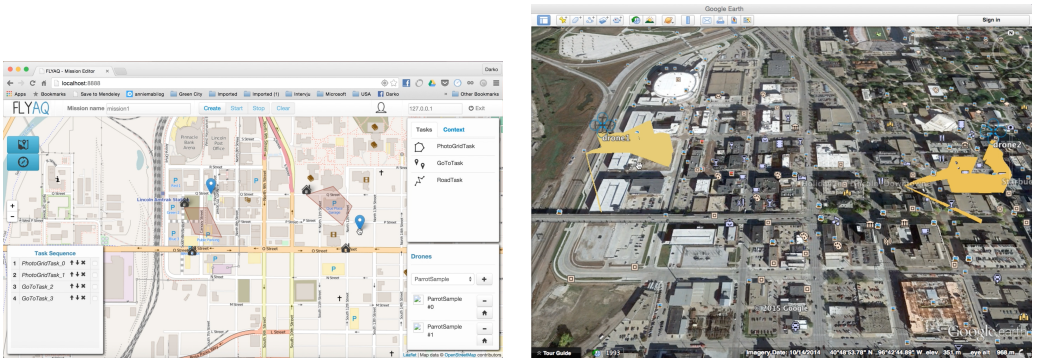
Basically, all the components of our simulation stack are open source and publicly available to the community. More specifically, the main component of our SITL simulation stack is MAVProxy[3], that is a developer-oriented, minimalist and extendable ground control station for any unmanned autonomous vehicle. We configured MAVProxy in order to use an already existing *drone physics simulator* that is able to simulate the physical and control characteristics of the well-known Arducopter[4]. Moreover, the DroneKit[5] module

[3]http://tridge.github.io/MAVProxy
[4]http://copter.ardupilot.com
[5]http://python.dronekit.io

within MAVProxy allows us to have programmatic access to the vehicles telemetry and, more importantly, to have programmatic control over the vehicles movements and operations. In light of this, we have been able to execute any BL model by (i) automatically transforming it into a Python script for each drone defined in the modelled mission via a model-to-text transformation and (ii) executing those Python scripts via the DroneKit APIs. Finally, we used the QGroundControl[6] ground station to (i) seamlessly collect all telemetry data (e.g., the drones geographical positions over time, their state, battery level, performed operations) via the MAVLink communication protocol, (ii) visualize it in an interactive graphical interface, and (iii) export it as commaseparated value textual files. In turn, exported telemetry files have been analysed via a set of Java programs we specifically developed for checking whether the simulated missions actually behave according to the properties of our automatic generation method (e.g., no collisions between the drones, no violation of no-pass zones, etc.). We exploited this simulation facility to test that generated models behave as expected. We considered missions involving 2 types of tasks (namely, for reaching a single specific geographical position and for taking photos across a grid of geographical positions) instantiated between 4 times per mission, involving 2 simulated drones and a variable number of contextual elements (mainly obstacles). One instance of our mission design and execution is presented on Figure 4.6 where we have two drones taking photos. When considering BL models, in our simulations we executed a maximum of 24 actions and 30 drone movements per mission. In order to further assess the feasibility of the approach, we also executed a full mission in a real setting with the Parrot AR.Drone 2.0 multicopter[7]; in this case the MML model of the performed mission is composed of one drone and four tasks, each one consisting in reaching a certain geographic coordinate and taking a photo with the front camera of the drone, whereas the generated BL model contains 6 actions and 9 movements. Interestingly, all the modelled missions have been fully executed by the platform without requiring to the operator neither any specific knowledge about the environment and the used drones, nor to write a single line of programming code. It is important to remark that the automatic generation has been conceived so to guarantee properties that are important for the degree of safety that is expected by users. The overall mission to be executed is split into parts, each part assigned to a drone, and movements and actions are performed so to realize the desired mission. The outcome of the generation activity can be seen as an ordered list of waypoints for each drone involved in the mission. Waypoints, as explained before, might have associated actions to be performed, and they are traversed by following a visit plan calculated so to avoid collisions with obstacles and other drones and to respect the restrictions imposed by no-fly zones. Our method assumes that information concerning obstacles and no-pass

[6] http://qgroundcontrol.org
[7] http://ardrone2.parrot.com

zones is complete and correct[8]. In other words, every existing obstacle and no-pass zone is correctly modeled.



(a) Mission Design (b) Mission Execution

FIGURE 4.6: Modeling missions for MMRSs

Synchronization between drones is realized through communication. It is important to note that we cannot have message loses since each drone communicates with the others by directly writing to the event queue of the receiving drones (the writing action is synchronous). Finally, it is important to note that complex interacting collective systems may expose emerging properties that represent unexpected behaviours that stem from interactions between the system parts and with the systems environment [70]. Emergent properties might be beneficial, but they can be also harmful, e.g., if they compromise the system safety. While considering critical systems it might be dangerous to accept and permit uncontrolled behaviours. Therefore, our idea is to limit as much as possible communications and interactions between drones. In particular, communications and interactions between drones are completely controlled and exclusively used for synchronizing drones before initiating new tasks. Any other form of communication between drones is blocked. This restriction will be relaxed in the next section where we describe the second part of our modeling framework that provides run-time elements that supports the mission execution in MMRSs.

4.4 Conclusions

In this chapter, our main objective is to provide a framework that makes the definition and realization of missions possible for people that are neither expert on ICT nor in robotics. In other words our framework (i) focuses on the definition of various tasks of a monitoring mission at an higher level of abstraction; (ii) allows engineers to automatically generate detailed flight plans from a user friendly, domain-specific, and graphical

[8]Dealing with uncertainly in this domain requires a dedicated approach; initial work might be found in [61, 69]

description of a mission; (iii) generates flight plans that avoid obstacles, collisions and no-pass zones; (iv) does not demand to manually specify each single waypoint of the mission (that actually may be hundreds in complex missions), rather it is able to automatically compute, plan, and assign all the waypoints that must be visited by each robot that is part of the MMRSs; and (v) is independent from the used task allocation, geometric and path finding algorithms, thus enabling for the use of state-of-the-art and well-established algorithms depending either on the next advances of those algorithms and on the traits of the missions to be performed in the future.

Chapter 5

Collective Run-time Adaptation for MMRSs

5.1 Introduction

Designing software for MMRSs in order to operate in unknown disaster scenarios is difficult. MMRSs need to be able to operate in uncontrollable and unknown environments, which are often shared with humans, and often they will be required to collaborate each other, and even with humans, to accomplish complex missions. Because of these challenges, these systems are both safety critical (e.g. failure or malfunction may serious injure people or create loss or severe damage to equipment/property) and mission critical (e.g. despite failures or malfunctions, the MRS should guarantee the accomplishment of the mission).

Possible scenarios where MMRSs operate may vary in many different aspects, like the environment, the scale of the affected area, the type of mission they need to perform. Environments in which they operate are often unpredictable and unknown; because of that MMRSs should be able to deal at run-time with unknown situations that cannot be anticipated completely at design-time. A model of the environment (containing for example obstacles, no-fly zones, wind and weather conditions) might be available, however, we cannot assume that such model will be always both correct and complete. The consideration of the environment when specifying the system comes due to the fact that a mission is always associated with a physical context where it is happening, so how a system will perform a mission strongly depends on the environment where it operates (e.g., the system will operate differently in environments with smooth vs. rough surface, environments with a lot of static obstacles vs environments with a small number of

obstacles etc). Hence, handling uncertainty upfront is often infeasible (or expensive). This means we need to deal with it when the knowledge becomes available at runtime.

In this chapter we will discuss about the second part of the modeling framework for supporting *execution* of MMRSs missions. A possible design-time phase for our execution framework can build on top of modeling framework for mission specification described in Chapter 4. The modeling framework for mission specification generates behaviour plans at design-time for each of the robots (specific types of agents) involved in the mission. The tool associates an agent (robot) a_i with a subregion SR for each of the mission tasks t. The correctness of the algorithms employed in the FLYAQ framework regarding preserving safety is proved in [71].

Here we focus on the run-time phase. In our framework, an adaptation is performed on-the-fly every time an unexpected system or environmental feature is observed in a part of the system. That being said, an adaptation is performed by one or a group of agents that are affected by it. Our framework supports a novel approach for managing the run-time adaptation of MMRSs. The approach is based on a generic collective adaptation engine that, operating at run-time, addresses collective adaptation problems in a decentralized fashion. New entities (in literature sometime referred as agents) can be introduced at any time during the mission execution in order to ensure the satisfaction of the mission. Once an adaptation issue is triggered, our approach provides a way to dynamically understand which parts of the system should be selected in order to solve such issue.

Chapter outline. The chapter is organized as follows. Section 5.2 describes our motivating scenario of a surveillance system in the premises of a company. In Section 5.3, we give an overview of the research challenges that appear in mission execution of MMRSs. Section 5.4 describes in details the execution framework and its parts. Section 5.5 presents an initial work on generic collective run-time adaptation in detail. Section 5.6 shows an initial evaluation of the approach in terms of feasibility, scalability, and survivability. Finally, related work to the area of this research is examined in Section 5.7 before the chapter is concluded in Section 5.8.

5.2 Motivating Scenario: a surveillance system

The motivating scenario of this paper is a surveillance system for the premises of a private company. The scenario will be also used as running example to explain details of the approach. The objective of the system is to detect an intruder that enters the

company's factory buildings and premises. The system involves various entities, each with specific roles. The entities involved in our scenario are the following:

Guards - persons that observe the surveillance process located in one of the buildings. When an intruder enters, a guard can physically approach him.
Maintainers - persons that are in charge of maintaining the equipment (i.e. drones, cameras, or sensors). They can make changes in the equipment: replacing old with new, adding new ones (e.g. add new drones or cameras), position the drones in their home position, etc.
UAVs - entities that follow specific protocols defined in the service agreement of the company. They can accomplish several tasks.
Movable Cameras - entities placed on the top of the building moving along its edges.
Fixed Sensors - entities placed in strategic places to monitor movements in the environment.
Ground Stations - entities that receive telemetry data and can do simple recalculations of the mission. A single drone can be managed by a single ground station, while one ground station can manage multiple drones.
Central Station - entity located somewhere in the cloud, where the surveillance videos are stored. Also, thanks to the central station guards can access video streams from a particular camera, sensor or drone in order to check a particular situation.

In normal operation mode, we have several drones patrolling the area, cameras moving along the edges of the buildings, fixed sensors located in strategic places sensing for a suspicious behaviour and a guard observing the whole process located in one of the buildings. If a suspicious behaviour is noted, the entities should perform as follows. First, one of the drones takes a picture of the intruder. Then, a drone delivers a warning voice message to the intruder regarding the actions that will be taken towards him. In parallel with this, notification is send to the guard about a suspicious behaviour.

Considering the scenario described above, the drones should be able to perform the following actions: (i) patrol the area searching for suspicious behaviour; (ii) one an intruder is found it will illuminate the area for a picture to be taken; (iii) take picture of the intruder; and finally (iv) communicate with the intruder by giving a warning message.

The number of entities in the system can vary according to the size of the area, number of buildings and other caracteristics of the specific company that wants to use the surveillance system.

5.3 Research Challenges

The principal entities of our scenario are quite autonomous and generally act independently. This makes the system highly dynamic. Indeed, from the point of view of a particular entity, the surrounding environment changes frequently and unpredictably and thus requires constant monitoring and adaptation.

Existing approaches (see Section 5.7) typically deal with multi-agent adaptive systems through isolated adaptation: each entity (agent) adapts itself independently from each other. However, in our scenario the problem is complicated by *collective behavior*. Indeed, even though the entities are generally autonomous, they dynamically form collaborative groups (we call them *ensembles*) to gain benefits that otherwise would not be possible. The example of such a collaborative group is an ensemble of drones, cameras and agents that cooperate for surveillance purposes: multiple entities must follow certain rules and in return the ensemble offers certain advantages with respect to having single entities working independently (e.g. robustness). Adherence to these collective rules temporarily reduces the flexibility of collaborating entities and has tremendous impact on how the entities adapt to dynamic changes. The isolated adaptation is not anymore effective. We can easily imagine what happens if there is a fault on a drone or on a camera: the ensemble as a whole needs to perform some collective adaptation in order to recover the broken devices (through the intervention of some maintainer) and to redistribute tasks to other entities in order to ensure the accomplishment of the mission. Even more serious consequences come if unexpected events will require an immediate and collective reaction to avoid collisions among drones; those collisions might compromise the safety of humans sharing the environment with the drones. This demonstrates that in adaptive systems with collective behavior new approaches to adaptation are needed: 1) *multiple* entities must adapt altogether and transactionally and 2) some kind of *negotiation* must take place to decide on the changes to be applied on each side.

Collective adaptation raises an important issue, i.e., identifying which parts of the system should be engaged in adaptation. This issue is not trivial at all, since a problem may be solved at different scale.

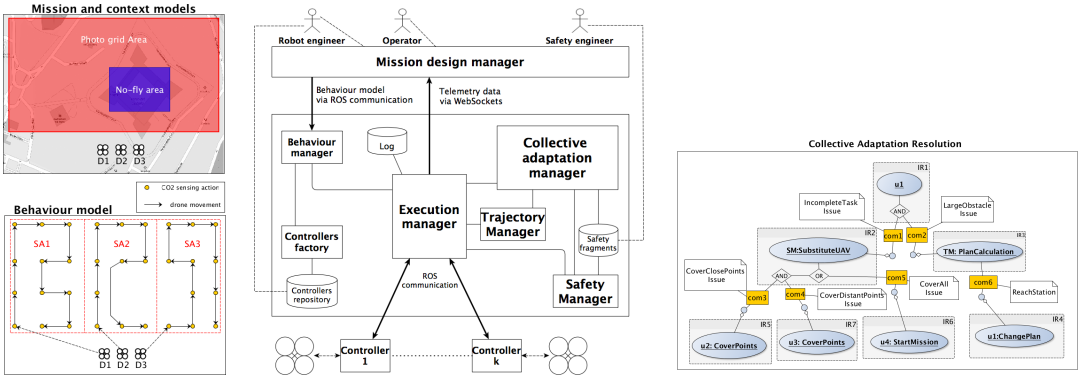
In the next sections we first provide an overview of a generic collective run-time adaptation approach for MMRSs and then we focus on the collective adaptation and issue regulation algorithm.

5.4 Framework for mission execution

As shown in Figure 5.1(b), the software architecture of our framework is composed of two main components: one component is dedicated to the design of the mission and the other one manages the execution of the mission at run-time. In this work we propose the second part of our framework for supporting **execution** of MMRSs missions. A possible design-time phase for our execution framework can build on top of the FLYAQ [3, 61] platform for the specification of missions of autonomous multicopters through a high-level and graphical domain specific language tailored to the specific application domains. In FLYAQ [3, 61] the tool generates behaviour plans at design-time for each of the robots (specific types of agents) involved in the mission. The tool associates an agent (robot) a_i with a subregion SR for each of the mission tasks t. The correctness of the algorithms employed in the FLYAQ framework regarding preserving safety is proved in [71].

Now, we focus on the run-time phase. During mission execution, at each point of time the system can follow the state of the mission. We define a mission state as follows.

Definition 5.1. A **Mission State** is a tuple $MS = (C, M, S, \tau)$ where: C is the context, M is the mission that should be performed and S is the MMRS system performing it at time τ.



(a) Models of the UAV mission (b) Software architecture of the approach (c) Issue resolution

FIGURE 5.1: Overview of the proposed approach

In our framework, an adaptation is performed on-the-fly every time an unexpected system or environmental feature is observed in a part of the system. We focus on the new self-adaptive features at run-time (lower part of the figure), while abstracting out the details about the mission design, which can be found in Chapter 4. When the *Behaviour manager* receives the behaviour model of the mission, it asks the *Controllers factory* to instantiate a pool of *Controllers*, each of them for each physical robot (e.g., a set of UAVs). Then, the behaviour manager triggers the *Execution manager*, which is in charge of (i) interacting with the controllers both to send their part of mission to be

executed and to receive telemetry data, (ii) checking when the safety-related conditions are violated in order to trigger adaptation-specific components, and (iii) to log mission data.

The adaptation performed by the *Collective Adaptation manager (CAM)* is based on the concepts of *entities* and *ensembles* [72]. Entities are basic building blocks representing the different actors and components of the system, e.g. robots, safety manager (SM), trajectory manager (TM), while ensembles (e.g., CO2 monitoring mission *C02M*) are a set of *roles* that can be played by participating entities. An ensemble role is specified with two main ingredients to manage collective adaptation: *issue* and *solver*. An issue is used to define a critical situation that can happen to a role of an ensemble (i.e., obstacle issue triggered by *u1* in Figure 5.1(c)), while a solver reflects the ability of a role to handle certain type of issue (i.e., *SubstituteUAV* of the *SM* in Figure 5.1(c)). To resolve an issue, the CAM uses a procedure that includes two activities: *issue resolution (IR)* performed internally at an ensemble role, and *issue communication (COM)* performed by a role when it asks other roles to resolve an issue. Figure 5.1(c) depicts the overall issue resolution executed internally to the *CO2M* ensemble. When the UAV *u1* identifies a large obstacle, it instantiates two issues: *IcompleteTask* and *LargeObstacle*. While in the first issue the other UAVs of the mission are strongly related, in the second issue they are not. To resolve the first issue, *u1* creates an issue resolution *IR1*. At this point the CAM component has to perform the issue communication *com1*. For that purpose, all ensemble partners are examined for the solvers that can resolve this type of issue. The *SubstituteDrone* solver of the *SM* builds an internal solution *IR2* by its internal adaptation procedure to manage the issue and the issues that it triggers. In this case the *SM* finds two possible solutions. In the first one, it will trigger two issues (*CoverClosePoints*, and *CoverDistantPoints*) that must be resolved simultaneously, whereas in the second one it will trigger only the *CoverAll* issue. The first solution involves *u2* and *u3*, which will enable the team to cover the points assigned to *u1*. In the second one, a recovery UAV *u4* will start the mission from scratch inheriting all the points not covered by *u1*. Finally, *u1* will fly back to the nearest ground station by the communication *com2* and with the help of the *TM* solver *PlanCalculation* that is able to calculate the trajectory that *u1* will travel to change its behaviour to internally resolving *IR4*. In the next section, we will describe this approach in more details.

5.5 Generic approach for collective run-time adaptation

In this section, we describe a novel approach for managing the run-time adaptation for MMRSs. The approach is based on a generic collective adaptation engine that, operating at run-time, addresses collective adaptation problems in a decentralized fashion. New entities (in literature sometime referred as agents) can be introduced at any time during the mission execution in order to ensure the satisfaction of the mission. Once an adaptation issue is triggered, our approach provides a way to dynamically understand which parts of the system should be selected in order to solve such issue. Our strategy is to organize the MMRS in different levels and to create a mechanism that decides the right scope for every adaptation issue. Our model of collective adaptation is built around the concept of *ensemble*, i.e. a collection of autonomous entities that collaborate to perform certain tasks.

5.5.1 Formal model for Collective Adaptation

In order to explain our model of collective adaptation we start introducing the notion of *entity*. An entity can be seen as a representation of a computational or human actor that can play multiple roles in different ensembles (i.e. a person entity can be a maintainer or a guard in our scenario).

Definition 5.2 (Entity)**.** An entity y is defined by a set of roles, R, it can play: $y = R$.

A role that an entity can play in an ensemble is primarily determined by the ways it collaborates with other roles. Collaboration consists in managing issues and responding to issues raised by others. As such, a role includes a set of issues it can produce, and a set of solvers it provides. *Issues* generally correspond to different critical situations that can happen to a role of an ensemble. Each issue includes a set of parameters describing it. An issue corresponds to a particular situation occurring in an ensemble.

Definition 5.3 (Issue)**.** An issue is a tuple $ui = \langle u, L_u \rangle$, where u is an issue type, and $L_u : u.uP \rightarrow V$ is an assignment function for issue parameters.

For example in our scenario the case of a drone it can trigger an issue type $obstacleDetected = \{obstacleDistance\}$.

Collaborating in an ensemble, each role can provide one or more solvers. When it is invoked to solve a specific issue it runs a dedicated solver defined as follows:

Definition 5.4 (Solver)**.** A solver is a tuple $s_i = \langle s, ui, L_s \rangle$, where: (i) s is the solver type; (ii) ui is the issue that the solver will solve and $ui.u \in s.U$; (iii) $L_s : s.uC \rightarrow V$ is

an assignment function for solver parameters and V are all possible values that solver parameters can be assigned with;

In our scenario each ground station is able to recalculate the path for some drones when for example some of them has identified an obstacle. To manage this situation it has a solver $RecalculatePath = \{\langle obstacleDetected, \{Battery\text{- }Level\}\rangle\}$ where $obstacleDetected$ represents the issue it is able to solve, while $BatteryLevel$ is a parameter that expresses the battery level of the drone involved.

A *role* is bound to a certain role type within an ensemble and its state is determined by its data (i.e. parameters and preferences) and by any ongoing issue resolution activities.

Definition 5.5 (Role)**.** A role is a tuple $ri = \langle ri_{id}, r, L_{ri}, P, \Uparrow_r \rangle$, where: (i) ri_{id} is the role identifier; (ii) r is a role type; (iii) $L_{ri} : rS \rightarrow V$ is a function that assigns values to the state parameters; (iv) P represents preferences of a role: $P = \langle riP, \mathbf{C} \rangle$, $riP \subseteq rP$ is a list of preferences of a role , $\mathbf{C}$ is a pairwise comparison matrix of the relative importance of pairs of preferences; (v) $\Uparrow_r$ is a set of active issue resolutions;

In the scenario, we can distinguish different roles: *D1*, *C1*, *G1*, *CS*, etc. To represent collaboration of multiple roles we introduce the notion of *ensemble*: it is a collection of autonomous entities that collaborate to perform certain tasks.

Definition 5.6 (Ensemble)**.** An ensemble is a tuple $ei = \langle ei_{id}, e, RI \rangle$, where ei_{id} is the ensemble identifier, e is an ensemble type and, RI is a set of role.

In our scenario the E1, E2, etc.. described in Figure 5.3 are examples of ensemble where $E1 = \{H1, D1, D2, C1, M1\}$, $E2 = \{H2, D3, D4, M2\}$, etc.

In our framework we have two types of activities that a role can execute during a collective adaptation problem resolution: *issue communication* and *issue resolution*. Issue communication is used to send an issue to a *target role* (see definition below) that is supposed to resolve it. The issue may be sent to multiple partners at a time in attempt to find a better solution. Issue communication comprises a few steps: 1) the issue is sent to all target roles; 2) the replies are received from the partners able to resolve the issue; 3) the preferable solution is chosen; 4) the preferable solution is committed. Formally, a target role and issue communication are defined as follows:

Definition 5.7 (Target Role)**.** A target role is a tuple $t = \langle ri_{id}, s_i, p \rangle$, where: (i) ri_{id} is the target identifier (role id); (ii) s_i is the solver invoked to solve the issue $s_i.ui$; (iii) p is the solution proposed by the target. It is a process that the target role will execute if it will become part of the overall issue resolution.

Definition 5.8 (Issue Communication)**.** An issue communication is a tuple $\uparrow_u = \langle ui, T \rangle$, where ui is an issue communicated, and T is a set of target roles;

While the issue communication is a way to propagate resolution activities between partners, *issue resolution* corresponds to the high-level model of internal elaboration being done by roles. In particular, we assume that the issue may either arise internally (when the issue originally occurs in this role) or is received by one of the role solvers. As soon as the issue is raised, the role may either resolve it locally or propagate issues to the other roles as a part of the resolution procedure. The issue resolution is formally described as follows:

Definition 5.9 (Issue Resolution)**.** An issue resolution is a tuple $\uparrow_r = \langle ri_{id}, ui, \Psi \rangle$, where: (i) ri_{id} is the identifier of the role, from which the issue arrived (null if the issue aroused internally); (ii) ui is an issue to be resolved; (iii) Ψ is a set of alternative solutions, each is a tuple $\psi = \langle ui, \Uparrow_u^O, p_{ext}, p_{int} \rangle$ where ui is the issue to be resolved, $\Uparrow_u^O$ is a set of outgoing issue communications, p_{ext} is a process (solution) that is supposed to be sent to the role associated with ri_{id}, while p_{int} is the internal process (solution) for the specific issue arrived.

If the resolution is fully local, in each solution $\psi \in \Psi$ the set of $\Uparrow_u^O$ of the outgoing issue communications is empty. Otherwise, $\Uparrow_u^O$ corresponds to the communication of all sub-issues that must be resolved in order to resolve the original issue.

The issue resolution procedure within an ensemble can be represented as a tree, which we call *issue resolution tree*. Indeed, the resolution procedure always starts from creating an issue resolution. It may instantiate further issue communications to resolve sub-issues. In turn, each issue communication may target a few roles, each of which consequently initiates an issue resolution and so on.

Definition 5.10 (Issue Resolution Tree)**.** An issue resolution tree is a tree $\mathcal{T} = \langle n_r, \mathcal{N}_r, \mathcal{N}_c, \mathcal{L} \rangle$ (where $\mathcal{N} = \mathcal{N}_r \cup \mathcal{N}_c$ are tree nodes made up by issue resolutions ($\mathcal{N}_r$) and issue communications ($\mathcal{N}_c$) such that $\mathcal{N}_r \cap \mathcal{N}_c = \emptyset$, $n_r \in \mathcal{N}$ is a tree root and $\mathcal{L} \subseteq \mathcal{N} \times \mathcal{N}$ are parent-child links between nodes) that has the following properties:

- $n_r \in \mathcal{N}_r$, i.e., root is issue resolution;
- $n \in \mathcal{N}_c \rightarrow \exists (n, l, n') \in \mathcal{L}$, i.e., leaves are always issue resolutions;
- $n \in \mathcal{N}_c \rightarrow \forall (n, l, n') \in \mathcal{L} : n' \in \mathcal{N}_r$ and $n \in \mathcal{N}_r \rightarrow \forall (n, l, n') \in \mathcal{L} : n' \in \mathcal{N}_c$, i.e., all children of an issue resolution are issue communications and all children of an issue communication are issue resolutions;

An issue resolution tree is a very intuitive abstraction for understanding and analysing how our approach works.

5.5.2 Collective Adaptation Process

In this work we assume that each entity implements the MAPE-K loop and more precisely implements the state machine in Figure 5.5. Moreover, the implementation of each entity should come with information about the solvers it provides and the issues it triggers. This information is given in terms of an XML file like the one in Figure 5.2. This excerpt contains part of the description of the Drone D1 and of one solver it provides.

```
<?xml version="1.0" encoding="UTF-8"?>
...
<!-- DRONE D1 -->
  <tns:role id="d1" type="D1">
    <tns:solver name="S11" selected="true">
      <tns:issue>
        <tns:issueType>I4</tns:issueType>
        ...
    </tns:issue>
       ...
      <tns:solution name="sol1">
        <tns:internalSolution>int1
            </tns:internalSolution>
        <tns:issue>
          <tns:issueType>I4.1</tns:issueType>
          <tns:issueCondition>tns:issueCondition
              </tns:issueCondition>
        </tns:issue>
          ...
      </tns:solution>
    </tns:solver>

    <!--Update planned trajectory (Local)-->
    <tns:solver name="S5" selected="true">
      <tns:issue>
        <tns:issueType>I10</tns:issueType>
          ...
      </tns:issue>
      <tns:solution name="sol5">
        <tns:internalSolution>int5
            </tns:internalSolution>
      </tns:solution>
    </tns:solver>
</tns:role>
```

FIGURE 5.2: Input file - description of the system

The solution involves various entities including UAVs, other devices, and also humans that collaborate as an ensemble. In the scenario described in Section 5.2 we make use of a UAV-based system to perform the surveillance of a big company. In the scenario we make use of the following entities: drones, i.e. multicopters, movable cameras, fixed sensors, humans, precisely guards and maintainers, ground stations and a central station. Each entity exposes specific roles within an ensemble and it can expose different roles in different ensembles; a role can produce issues and provide solutions.

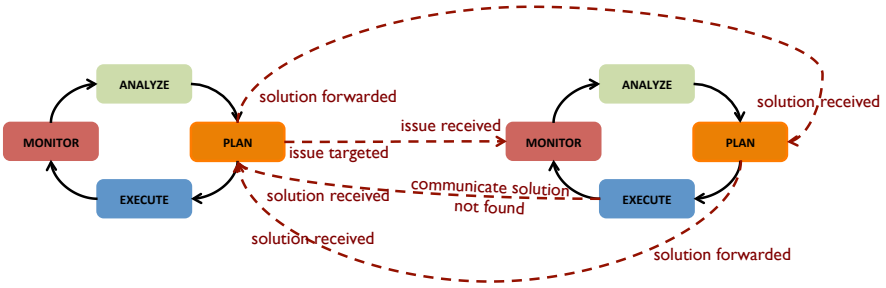


FIGURE 5.3: Overview of the communication between two entities, issue triggered by one in the left hand-side

Figure 5.3 shows the communication between two single entities, only presenting the possible flow of information when an issue is raised by the entity in the left-hand side. This is a simplification for presentation purposes with the aim to show the type of communication and synchronization between entities. We have to consider that in normal scenarios our framework is able to deal with multi-entities that all together collaborate to solve issues and each entity can raise new issues. For instance in order to complete Figure 5.3 we should add the entire flow of communication triggered by the entity in right-hand side. The labels in the dashed arrows represent the computational state of the respective state machines. The states of the state machine that can send or receive messages from other entities are identified by a dashed arrows. Focusing on the entity on the left hand-side of Figure 5.3, during the planning phase and more precisely in the Issue_targeted state (see the state machine in Figure 5.5), the entity triggers an issue that is captured by the entity in the right-hand side in the monitoring phase and more precisely in the state issue_received. Then the entity in the right-hand side can replay in two ways: (i) solution found, from state solution_forwarded in the planning phase, or (ii) solution not found, from state communicate solution not_found in the execute phase.

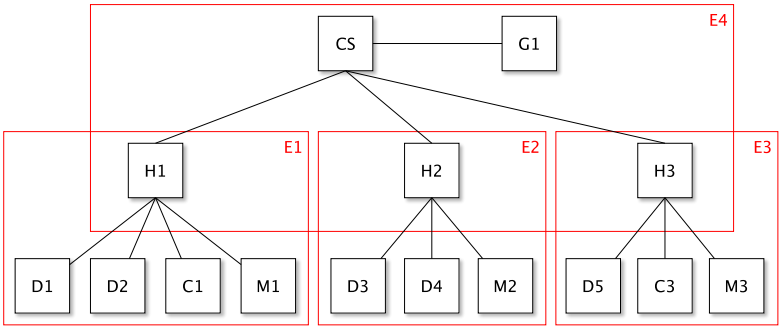


FIGURE 5.4: Types of agents and communication topology

Entities might be organized in several possible topologies according to the needs of the particular domain. Figure 5.4 shows the topology of the example. It is hierarchical with

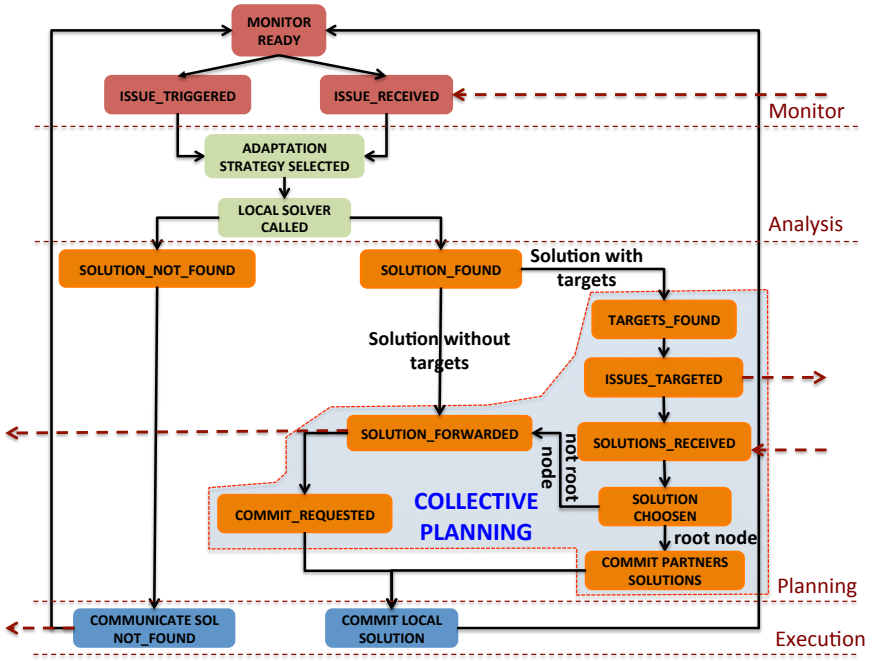


FIGURE 5.5: State machine animating each agent of the MAPE loop hierarchy

the central station, i.e. CS, coordinating the other entities. CS together with his sub-entities, i.e. three group stations, named $H1$, $H2$, and $H3$, and a guard (human) called $G1$, forms the $E4$ ensemble. $H1$ manages the ensemble $E1$ that includes also drones $D1$ and $D2$, camera $C1$ and the maintainer $M1$. $H2$ forms the ensemble $E2$ together with drones $D3$ and $D4$ and maintainer $M2$. $H3$ manages the ensemble $E3$ that includes also drone $D5$, camera $C3$ and the maintainer $M3$.

Figure 5.5 shows the state machine that each entity should implement. The states belonging to the monitoring phase are the MONITOR READY state, representing the situation in which the entity is monitoring its environment, and two more states related to the issue triggering that are: the ISSUE_TRIGGERED state, which is reached when an issue is triggered internally to the entity, and the ISSUE_RECEIVED state, which is reached when the entity is invoked by another entity in the ensemble for solving an external issue.

In both cases the analysis phase is started. It is composed by two states, namely the ADAPTATION STRATEGY SELECTED state and the LOCAL SOLVER CALLED state, to select the right adaptation strategy and to run the corresponding solver, respectively. If the solver is not able to found a solution, the state machine reaches the COMMUNICATE SOL NOT_FOUND state (triggering the Execution phase); this represents the situation in which the failure is forwarded to the entity that originally generated the issue, in particular to its monitor in order to restart it and/or to allow it to generate other issues. If the selected solver found a solution the state machine enters

in the COLLECTIVE PLANNING phase, where a solution is found collectively with all the entities involved in the issue resolution.

Specifically, if the solution previously generated by a solver does not include the involvement of other entities in the ensemble, then the SOLUTION_FORWARDED state is reached. In this state, the entity simply sends information (e.g. fitness value) about its capability to solve the current issue and it waits for a request of commit that will further move the state machine to the COMMIT_REQUESTED state.

On the contrary, if the solution includes the need to involve other entities in the resolution process, the state machine moves to the TARGETS_FOUND state, in which external dependencies are derived, and soon after, in the ISSUES_TARGETED state in which the entities involved in the resolution are triggered. At this point the entity that has started the collective planning, receives the feedback by the other entities and its state machine moves to the SOLUTION_RECEIVED state. From this feedback it derives the final solution and it moves to the SOLUTION CHOOSEN state.

As soon as all preliminary steps have been done, the collective solution is generated. Here we should distinguish two cases. If the issue was triggered internally (root node), the entity first commits all dependent solutions, by moving in the state COMMIT PARTNERS SOLUTIONS and then it commits its local solution reaching the state COMMIT LOCAL SOLUTION. Instead, in the case in which the original issue was coming from outside, the entity reports the feedback to the issue's sender while moving in the state SOLUTION_FORWARDED, and then it waits for a future commit in the state COMMIT_REQUESTED. Every time that an entity is involved in the collective planning phase, and it is waiting in the COMMIT_REQUESTED state, it can receive a positive or a negative reply for the proposed solution. In both cases, it executes a solution commit, which results to be empty in the negative case, and it eventually reaches the COMMIT LOCAL SOLUTION state.

5.5.3 Issue Resolution Algorithm

To realize our approach, in Figure 5.6 we abstractly define an algorithm that covers the procedures for issue resolution and commitment (functions resolve and commit respectively). The function resolve is used recursively to trigger a distributed resolution procedure across multiple entities within an ensemble.
The function is called locally by an entity e that originally detected an issue i. Further recursive calls are propagated using a *Remote Procedure Call* (line 10). The function includes the following important steps:

```
1  function resolve(i,e)
2      sol := callSolver(i)
3      foreach s.issues ∈ sol:
4          Com := derive_coms(s.issues)
5          foreach c ∈ Com
6              Target := find_targets(c)
7              S = ∅
8              s_best = null
9              foreach t ∈ Target
10                 t.solution = rpc(resolve(c.issue,t))
11                 S = S ∪ t.solution
12             s_best := AHP(S)
13     if e != root
14         store(s_best)
15     else
16         commit(s_best, Target)
17
18 function commit(s_best, Target)
19     foreach t ∈ Target
20         execute(t.best)
21     execute(s_best)
```

FIGURE 5.6: Issues Resolution Algorithm

Line 2. The solver of the entity e is invoked and a solution *sol* is calculated for the specific issue i. Function callSolver is beyond the scope of this paper but may generally exploit various entity-specific and domain-specific solvers.

Lines 4-11. For the solution returned, a set of sub-issues is identified. This is done with the function derive_coms that derives all sub-issues that must be resolved for a given solution in form of corresponding issue communications. For each issue communication, the set of potential solvers is identified across all reachable entities (function find_targets). Finally, to understand how well the targets can handle sub-issues, the resolve function is called remotely on the targets (line 10) and all the possible solutions are memorized in S.

Line 12. Once the solutions to sub-issues are obtained from the remote entities, the Analytic Hierarchy Process (AHP) algorithm [73] is executed to identify the best solution s_{best}.

Lines 13-17. If the current entity is not the resolution tree root, the best solution is stored locally (function store). If the current entity is the resolution tree root (line 16), commit is executed. Function commit (line 18) enacts a *distributed commit* of the best solution. It takes as input the best solution and the set of involved entities in the specific issue resolution. It includes the following steps:

Lines 19-20. *Execute* is called for each of the target entities corresponding to the best solution. Execute is an asynchronous function, so as not to impede the solution execution on line 21.

Line 21. Entity e executes the internal solution corresponding to the best solution.

5.5.4 Application to the surveillance scenario

According to the scenario described in Section 5.2, during the system's normal execution, the following issues may be raised.
(I1)Equipment fault detected - Entities that can raise this issue are: (i) cameras, (ii) drones, (iii) sensors, (iv) ground station.
(I2) Possible Collision between drones - Entities that can raise this issue are: drones and ground station
(I3) Low Battery - Entities that can raise this issue are: cameras, drones, sensors, ground station
(I4) Intrusion detected - Entities that can raise this issue are: cameras, drones, sensors. This is a general issue. For a resolution, more specific sub-issues should be defined. For each of the actions defined in the service agreement a sub-issue is raised. From our scenario defined above, we can define I4 as a set of the following 4 sub-issues: *(I4.1)* Light the area where the intruder is detected; *(I4.2)* Photo of the area should be taken; *(I4.3)* Notification to the guard should be send; *(I4.4)* Warning message to the intruder should be send
(I5) Obstacles detected - Entities that can raise this issue are: drones and ground station.
(I6) Mission incomplete - This issue can be raised as a consequence/result of the previous six issues.

For each of the issues mentioned above, the entities provide specific solvers that solve a specific issue. In our scenario we have the following solvers.
(S1) Stop an entity - provided by maintainer and guard
(S2) Start new entity - provided by maintainer and guard
(S3) Manually guide the drone - provided by guard
(S4) Recalculate path - provided by ground station and central station.
(S5) Update planned trajectory - provided by drones
(S6) Recalculate mission - provided by the central station
(S7) Illuminate the area - provided by drones
(S8) Take a photo - provided by drones
(S9) Communicate with intruder (send voice message) - provided by drones and guard
(S10) Send notification to guard - provided by the central station
(S11) Replace equipment - provided by the maintainer

Here we will describe how the resolution happens. We will go through each of the issues and we will describe which solvers are appropriate for the different issues.

I1: When some of the entities (camera, sensor, drone, ground station) have a fault, the issue **Fault detected** is raised. In parallel, the issue **Mission incomplete (I6)** is raised as well. Here, the maintainer can solve the first issue (I1) by providing the solver S1 (stopping the entity).

I2: In parallel with this issue, the issue **Mission incomplete (I6)** may be raised. In order to resolve possible collision between drones we have three possible resolutions. First, the drone itself can provide solution - S5. Second, the ground station or the central station can recalculate the path of the drones that are included in the possible collision - S4. Depending on the computation that needs to be performed (central station has higher computation power) one of the solutions is chosen. Third, the guard can manually guide the drones - S3. In choosing the solution important role plays the distance between the drones and their speed. If there is enough time, central solution can provide the solution. If that is not the case and if we need prompt solution, we make simple change in the directions of the drones(both drones turn left) or the guard manually guides the drones to a safe position. If it is something in between we can choose the solution provided by the ground station.

I3: When some of the entities (camera, sensor, drone) have a low battery, the issue **Low battery** is raised. In parallel with that, the issue **Mission incomplete (I6)** is raised as well. Here, the maintainer can solve the issue I3 by providing S1 (stopping the entity).

I4: In parallel with this issue, the issue **Mission incomplete (I6)** may be raised. In order to resolve the general issue **Intruder detected** we use the formal service agreement mentioned above. For each of the actions in the agreement a sub-issue was raised. The resolution asks each of the sub-issues defined above to be resolved. In our case, I4.1 is resolved by S7, I4.2 is resolved by S8, I4.3 is resolved by S10 and I4.4 is resolved by S9. When taking in consideration the solutions between different drones we put attention to the position of the intruder, the drone's position and the position of the safety zone. Safety zone plays important role, because solution that requires from the drones to cross the safety zone have lower priority comparing to the other solutions.

I5: In parallel with the issue **Obstacles detected**, the issue **Mission incomplete (I6)** may be raised. In order to avoid the obstacles we have the following resolutions. First, the drone itself can provide solution - S5. Second, the ground station can recalculate the path taking in consideration the size of the obstacle - S4 and third, the guard can manually guide the drone - S3. In choosing the most appropriate solution the distance between the drone and the obstacle and the drone's speed play important role.

I6: To resolve this issue, which appears as a result of the previous issues we need first to implement S11. After that we use the solver S6 and in the end S2. In that point, we replaced/added an entity, we did recalculation of the mission and we started the new

entity. If the issue is still not resolved, we repeat the cycle with solvers until resolution is achieved.

In the next section we show the experimental evaluation we performed on the scenario described above.

5.6 Evaluation

We implemented the first version of the CAE in Java and we evaluated it by executing an experiment with a focus on how it performs in terms of feasibility and scalability. In our experiment we are concerned with measuring and understanding how the CAE is able to provide solutions in terms of collective adaptation strategies. In order to automatically solve an issue at the level of each entity, we adopt service composition approach presented in [74]. According to it, an issue is transformed into a planning problem and planning techniques are used to resolve it. For this reason we do not focus the attention in the correctness of the solution provided by each solver but instead in the evaluation of our collective adaptation approach.

In the following we present the detailed experiment design and the discussion of its results.

5.6.1 Experiment Design

Goal. In order to ensure that important aspects of our experiment are defined before executing it [16], in Table 5.1 we formulate the goal of our experiment by using the Goal-Question-Metric perspectives [39].

Analyze	the Collective Adaptation Engine (CAE)
for the purpose of	evaluation
with respect to	its feasibility and scalability
from the point of view of	a researcher
in the context of	UAV-based missions and the current CAE implementation.

Table 5.1: Goal of our experiments

Research questions. The goal presented above can be refined into the following research questions:

RQ1: Can the Collective Adaptation Engine (CAE) be used at run-time to manage the adaptation of UAV-based missions?

RQ2: Is the CAE scalable for managing real-sized missions performed by real UAVs?

The objective of these research questions is to measure and understand how the CAE is able to support run-time collective adaptation. It is important to note that we do not aim at observing the actual execution of the identified solutions (e.g., analysing the exact adapted trajectories, how obstacles are actually avoided, how photos are taken, etc.); this choice is based on the need to clearly and sharply isolate the measurements of the CAE from all the potential sources of confusion and bias coming from the actual execution of the missions. Experimentation on the full execution of real missions is in the focus of another work we are currently working on.

Context. The experiment is designed as a multi-test within object study [16], because it is conducted on a single *object* (i.e., the current implementation of CAE) across a set of *subjects* (i.e., a set of UAV-based missions).

Independent variables. In the following we provide the independent variables of our experiment:

iv_1: number of issues raised while executing the mission;

iv_2: the machine in which the CAE is deployed and running, it can have two values:

- M: a dedicated machine running an Ubuntu 14 LTS Linux distribution with a 32-cores (each of them with a 1.4Ghz frequency) and 64Gb RAM;
- R: a Raspberry Pi 2 Model B[1] running a Raspbian 7 Linux distribution with a 900MHz quad-core ARM Cortex-A7 CPU, and 1Gb RAM.

The main idea of running our experiment on two different machines is to assess whether the current CAE implementation can be used for run-time adaptation both on highly performing and less powerful machines. In our scenario, the central and ground stations can be represented by the highly performing machine M of the experiment, whereas the drones D1-D5 can be represented by the less powerful Raspberry Pi device R of our experiment.

Dependent variables. In the following we provide the dependent variables of our experiment, together with the research questions they are addressing:

[1] https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

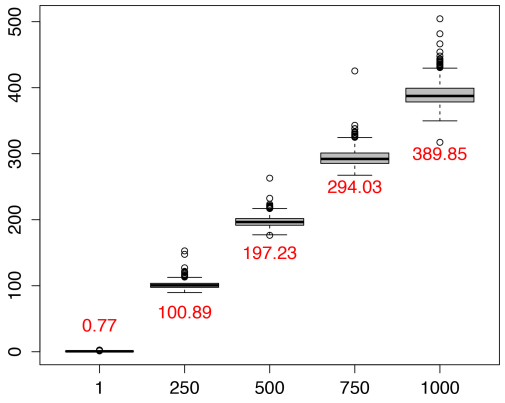


FIGURE 5.7: Execution time on M per number of raised issues

dv_1: execution time of the whole adaptation part of the mission in milliseconds (RQ1);

dv_2: the amount of memory used for managing all the raised issues during the mission (RQ2);

Subjects selection. In our experiment we perform a stratified random sampling [16]: the population of all possible sequences of raised issues is divided into a set of groups with a uniform distribution between the groups in terms of the number of raised issues. Random sampling is then applied within the groups. Identified groups are those in which the number of raised issues is one of the following: $< 1, 10, 100, 200, 500, 1000 >$.

Treatments. Each treatment models a sequence of raised issues within a mission. Each treatment is represented by a tuple $< iv_1, iv_2 >$, where we assign specific values to each independent variable within the tuple. More specifically, in each treatment iv_1 can have one of the following values: $< 1, 250, 500, 750, 1000 >$, iv_2 can be either M or R. For what concerns the types of issues, we raise issues of type I1, I2, I3, I4, and I5; we do not consider I6 issues because they are raised as a result of one of the other I1-I5 previous issues. Within each treatment, both the order of the raised issues within the sequence and the entities raising them is randomly chosen. As an example, the values $< 100, P >$ represent the treatment in which the total number of raised issues is 100, where 30 issues can be of type I1, 20 issues can be of type I2, etc. (summing up to 100 issues in total), and the experiment has been run on the Raspberry Pi device.

5.6.2 Discussion of Results

We created 1000 treatments for each single combination of all the possible values of our iv_1 and iv_2 independent variables, resulting in a total of $1000 \cdot (5 \cdot 2) = 5000$ runs of the experiment.

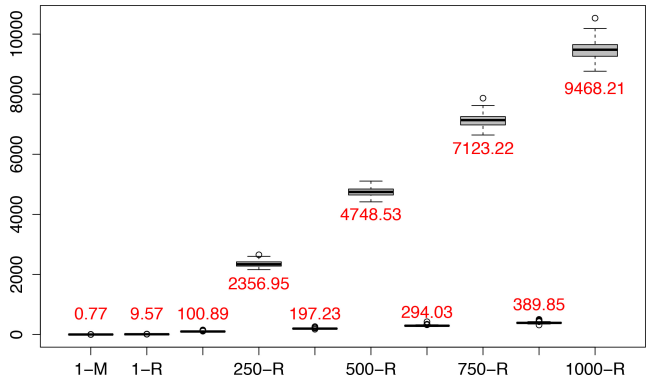


FIGURE 5.8: Execution times per number of raised issues (in milliseconds)

For what concerns the **use at run-time** of the CAE to manage the adaptation of UAV-based missions (RQ1), we measured the execution times of the mission represented by each treatment. Figure 5.7 shows the distribution of execution times for each value we assigned to iv_1 (for each column the numbers in red represent the mean of all the values obtained for each class of treatments). More specifically, in Figure 5.7 we show how the execution time varies when the current CAE implementation is deployed and running on the highly performant machine M. The first observation is that the execution time of whole treatment has a linear trend with respect to the number of raised issues; this is something we could expect since here we are measuring the total amount of the time that the whole treatment takes to complete. More interestingly, we can notice that solving a single issue on M takes in average 0.77 milliseconds, whereas it takes an average of 9.57 milliseconds on the Raspberry Pi device (see Figure 5.8). Also, we can observe that solving a full sequence of 1000 issues on M takes an average of 389.85 milliseconds; this result is encouraging since it shows that the current implementation of CAE can solve a burst of 1000 issues in under 400 milliseconds[2]. As expected, when running on the Raspberry Pi, our CAE implementation performs worse, but still it gives highly encouraging results with an average of 9.57 milliseconds for solving a single issue (together with its sub-issues according to the resolution tree) and a maximum resolution time of 9468.21 milliseconds for solving a burst of 1000 issues. By looking at the size of the boxes in the figure we can see another interesting result, indeed the execution times of our CAE implementation has a very low variance with respect to the number and type of raised issues, meaning that our CAE is able to produce adaptation plans within a predictable (low) amount of time, thus making us reasonably confident about the possibility of using it at run-time to manage the adaptation of UAV-based missions.

For what concerns the **scalability** of the CAE for managing real-sized missions performed by real UAVs (RQ2), we decided to measure the memory consumption of the

[2]It is important to consider that having a burst of 1000 issues all together is very rare in real missions.

whole issue resolution algorithm. Figure 5.9 shows the memory used in the JVM heap for storing all the Java objects needed during the whole mission.

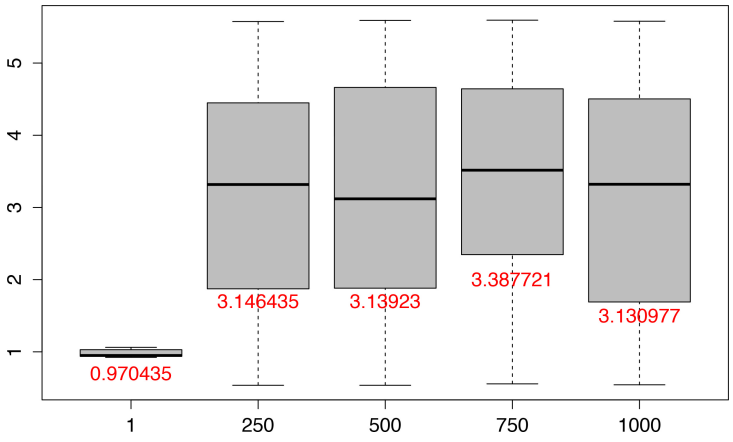


FIGURE 5.9: Memory consumption on the Raspberry Pi per number of raised issues (in Mb)

The obtained results tell us that for solving a single issue our CAE uses an average of 0.97 Mb of memory, whereas for solving a burst of 1000 issues to be solved all together (which we recall is an extremely pessimistic situation) it uses an average of 3.13 Mb of memory. As we could expect, the amount of used memory is nearly constant, independently of the number of raised issues (e.g., 250, 500, 750 issues) because the Java garbage collector is able to reclaim unused memory from the JVM heap making it available for subsequent computations. This behaviour of the JVM also explains the higher variance that we can notice when considering the memory consumption during our experiments with respect to the low variance we saw regarding the execution times; indeed, in Java the developer cannot fully control the exact instant in which the garbage collector must be executed, rather the System.gc() instruction is a simple *suggestion* to the JVM that does not give any assurance about the exact instant in which the garbage collector is actually executed. Finally, the obtained result about memory usage gives a clear indication about the capability of our implemented CAE to scale for managing real-sized missions. Indeed, we need to recall that many open-source and commercially available UAVs are equipped with far more RAM memory than the one needed in our worst case scenario. For example, the most basic entry-level UAV called AR Drone Parrot is equipped with 1Gb of DDR2 RAM memory. Also, as discussed in Section 5.5, the resolution algorithm is also deployed on much more powerful machines, like ground stations and the central station, which do not suffer from stringent hardware limitations like UAVs do.

Replication package. To allow easy replication and verification of our experiment we provide to interested researchers a complete and portable replication package. The replication package is publicly available[3] and contains all the Java code, R scripts, and obtained data of the experiment.

5.7 Related Work

Work	Agent Type		Behaviour		Coalition Management		Utility		Hierarchy		Adaptation	
	Homog.	Heter.	Selfish	Coop	Centralized	Decentralized	Yes	No	Yes	No	Design	Run-time
[75]	✓			✓	✓		✓			✓		
[76]	✓			✓	✓		✓			✓		✓
[77]	✓			✓	✓		✓			✓	✓	
[78]	✓			✓			✓			✓		✓
[79]		✓		✓	✓			✓		✓		✓
[80]		✓			✓	✓		✓				✓
[81]		✓			✓	✓		✓			✓	
[82]		✓		✓		✓	✓		✓		✓	
[83]		✓		✓					✓		✓	
[84]		✓		✓				✓		✓		✓
[85]		✓		✓	✓		✓		✓			✓
[86]		✓		✓		✓	✓			✓		
[87]		✓		✓		✓		✓		✓		
[88]		✓		✓		✓		✓	✓			
[89]	✓			✓		✓	✓			✓		
[90]	✓			✓		✓	✓			✓		✓
[91]	✓			✓		✓			✓		✓	
[92]		✓		✓		✓	✓			✓	✓	
[93]		✓		✓	✓		✓		✓			✓

TABLE 5.2: Literature Comparison

In this section we will review recent works on coalition formation for multi-agent systems, on multi-party sessions and choreographies, on component ensembles and on run-time modelling for mobile multi-robot systems. In Table 5.2 we review these works and categorize them according to the following criteria:

1. **Entity type**: whether the considered agents/entities are homogeneous or heterogeneous. It does not simply distinguish whether agents can have different roles in a coalition, but also indicates whether the proposed method allows for different types of role or just for a *rigid* structure of the coalition.

2. **Behaviour**: whether the agent acts in a selfish or cooperative way.

3. **Coalition management**: whether coalition is managed by a coalition manager or in a distributed way among the agents.

4. **Utility**: whether the decisions of agents depends on some non functional calculation. It gives information whether the decision making mechanism of an agent is based on some sort of metrics.

[3] http://cs.gssi.infn.it/seams2016

5. **Hierarchy**: whether the coalition formation mechanism allows a hierarchical structure of them, more specifically gives information whether the approach can deal with systems of systems (e.g. a coalition of coalitions).

6. **Adaptation**: whether the approach deals with adaptation and if yes, can the approach be used only at design-time or also at run-time.

Coalition formation has been widely studied in game theory and economics. In multi-agents system, and more in general in Artificial Intelligence (AI), coalition formation has been used as a mean of dynamically creating partnerships or teams of cooperating agents. Many works on coalition formation in multi-agents systems, e.g. [94, 95], use the assumption that all agents can directly communicate with each other, which is not realistic in the real world. [75] tackles the problem of coalition formation in multi-agents systems in a neighbourhood agent network (a network in which agents communicate directly only with their neighbours). Agents can participate to several coalition at the same time, by indicating for each of them the *degree of involvement* (DoI).

Multi agents systems have been applied in variety of applications ranging from the field of electricity markets [76, 77] to supply chains management [96] and complex software systems building [97]. Here, we take in consideration solutions proposed in different areas.

Regarding the first criteria of agent type, we have few solutions which are application specific i.e. we regard them as homogeneous agents (ex. [76], [75], [77] etc.) while most of the related work is more general and it includes agents with different types of roles i.e. heterogeneous agents (ex. [81], [82], [83]).

Regarding the second criteria, all the works include solutions where the agents cooperate to achieve a specific common goal and no study was found which regards the selfish behaviour of the agents.

Regarding the third criteria Coalition Management, we have few studies where in the solution there is designated a centralized coordinator to collect information from all agents and then to disseminate a decision to the whole group. However, such a strategy interferes with the system's scalability and robustness: the coordinator can easily become a communication bottleneck, and it is also a potential point of failure for the system. Because of that, most of the works mentioned here propose decentralized approach. For example, in [76] is presented a decentralized and dynamic method where coalition formation is achieved by opportunistic aggregation of agents, while maximizing coalition benefits by means of taking advantage of local resources in the grid.

Regarding the fourth criteria, we have a combination of studies which provide solutions that contain some utility function where agent coalition takes place considering the resources and the performance of the system and studies where solutions don't provide that. For example, in [78] is presented an incremental approach to self-organization based on *bottom-up* coalition formation. Here, agents negotiate to maximize the system's global utility by using a variety of protocols based on local or social marginal utility. Through coalition formation, agents in a large system faced with a set of tasks partition themselves to maximize system performance.

Concerning the fifth criteria, we have only few studies where the approaches take in consideration the hierarchy of the system, thus contributing to scalability. Example of a coalition formation mechanism which allows a hierarchical structure is presented in [83]. The approach is named Helena - Handling massively distributed systems with ELaborate ENsemble Architectures and it represents a modelling technique centered around the notion of roles teaming up in ensembles. Ensembles are built on top of a component-based platform as goal-oriented communication groups of components. The functionality of each group is described in terms of roles which a component may dynamically adopt. Other theoretical approach presented in [93] provides a suitable basis for reasoning about reconfigurable multi-agent systems by building a hierarchy of agents according to their responsibilities in achieving certain kind of goals. The formalisation covers the notions of system goals and agents, various formal structures (functions and relations) defining different interrelationships between these notions, as well as constraints on the system dynamics allowing a multi-agent system to become more reconfigurable. The final result are the established connections between goals at different levels of abstraction, system architecture and agent responsibilities.

Concerning the sixth criteria, we have some studies which discuss about approaches which can deal with adaptation. In order to deal with increasingly growing complexity of the mission-critical software systems, there are some recent works which take consideration adaptation at runtime. Contemporary mission-critical software systems are often expected to safely adapt to changes in their execution environment where run-time adaptation mechanisms reduce the complexity of the system. For example, in [84] is presented an adaptive run-time model used to establish a flexible information processing within a group of heterogeneous robots, while in [85] is presented a reusable framework for developing adaptive multi-robotic systems for heterogeneous robot teams using an organization based approach. The framework is based on the Organizational Model for Adaptive Computational Systems (OMACS) and the Goal Model for Dynamic Systems (GMoDS). OMACS is an abstract model used to capture the system configuration and allows the team to organize and reorganize without the need for explicit runtime reorganization rules. The framework is based on runtime models for understanding (1) what

the system should be doing in terms of system goals and (2) how the system is organized to achieve these goals.

All the work discussed previously focus on one or few criterion we defined above. Contrariwise, our approach is general, meaning that it is not domain or problem specific. This gives us the opportunity to reuse it in different domains. In this particular work, the approach was adjusted in the domain of UAV based systems. In contrast to related works, our approach gives us the possibility to handle both cooperative and selfish behaviour between agents. Furthermore, the management of coalition is decentralized and this eliminates the single point of failure and the potential bottleneck in the system. Moreover, as we can see in the example, our approach provides support for a hierarchical structure. This is very useful because it provides scalability and allows different agents with different knowledge to take decision at different levels. And in the end, as we can see from the example, our approach provides support for run-time adaptation in mission critical and safety critical systems. In summary, comparing to the other works, which focus only on one or few criterion, we cover all the aspects in our classification framework.

5.8 Conclusions

In this chapter we presented the second part of our modeling framework that supports execution of MMRSs missions. Furthermore, we presented an approach for managing the run-time adaptation of MMRSs. The generic adaptation engine that we use identifies the strategy to be used in order to orchestrate the various solvers offered by the entities that compose the MMR system. The approach has been empirically evaluated through a large experiment in the context of a private company surveillance scenario.

Chapter 6

Managing Safety and Mission Completion via Collective Run-time Adaptation

In Chapter 5, we provided a generic approach for managing run-time adaptation with general types of problems and solvers that can be triggered during mission execution. In this chapter we will discuss how our framework manages and resolves specific problems which strongly depends on the specification of the mission and its domain. We present two adaptation resolution methods: one for (potentially partial) resolution of mission problems and one for full resolution of safety problems.

Chapter outline. The chapter is organized as follows. Section 6.1 describes our motivating scenario of a carbon dioxide monitoring system on which we base our approach. In Section 6.2 we provide background notions for setting the context of our execution framework by discussing modular reusable behaviours execution, followed by a description of the Behaviour Tree Architecture as an architecture for modeling generated behaviour plans. Section 6.3 describes in details the execution framework and its corresponding parts, while Section 6.4 defines a formal model for the problem resolution process. Section 6.5 presents our mission problem resolution process for (potentially partial) satisfaction of mission objectives, while Section 6.6 shows our safety problem resolution process for full satisfaction of safety objectives. Finally, related work to the area of this research is examined in Section 6.7 before the chapter is concluded in Section 6.8.

6.1 Motivating Scenario

In this section, we describe a motivating scenario of a carbon dioxide monitoring system (Figure 6.1). This scenario will be used as a running example to explain details of the approach. Figure 6.1 shows a mission in which three robots, specifically three Unmanned Autonomous Vehicles (UAVs), have to monitor the CO2 levels within a geographical area; the team of UAVs has to sense the CO2 level of each geographical point in a grid composed of cells of size $10x10$ meters. The mission is considered as successfully completed if the whole area has been fully monitored. Starting from this very high-level description of the mission, the configurations and flight plans for the UAVs can be automatically generated. Once configured, these UAVs perform the mission by flying from their initial position to the border of the monitoring area. Then, each UAV starts monitoring a specific sub-area so that the whole team can cover the entire area in parallel. A sub-area can be decomposed in number of blocks. A block is the smallest measurable unit for the mission region. Each block is assigned a unique identifier. We represent the size of an area using the number of blocks along each dimension.

The bottom side of Figure 6.1 shows the mission execution of three UAVs. Each UAV is executing its corresponding behaviour to cover its part of the mission. The green region is the region that has been monitored by the corresponding UAVs. The blue region represents the region that should be monitored by the blue UAV, the yellow region is the region that should be monitored by the yellow UAV and the purple region is the region that should be monitored by the purple UAV.

Let us assume that a UAV u_x (the blue UAV on Figure 6.1) of the team U must reach a target geographical position p and it identifies an obstacle along its trajectory towards p; if the obstacle is avoidable (e.g., a tree), then u_x adapts its trajectory so that it avoids the obstacle to reach p; if the obstacle cannot be easily avoided (e.g., the large gray object on Figure 6.1, then the behaviours of u_x and some other UAVs in U are adapted so that the position p is still covered by another UAV $u_i \in U$ and u_x can cover some other points within the area.

6.2 Modeling modular behaviours

6.2.1 Modular behaviours

In FLYAQ [3, 61], at design-time safe behaviour plans are generated for the agents involved in the mission according to the initial set of active agents and the mission

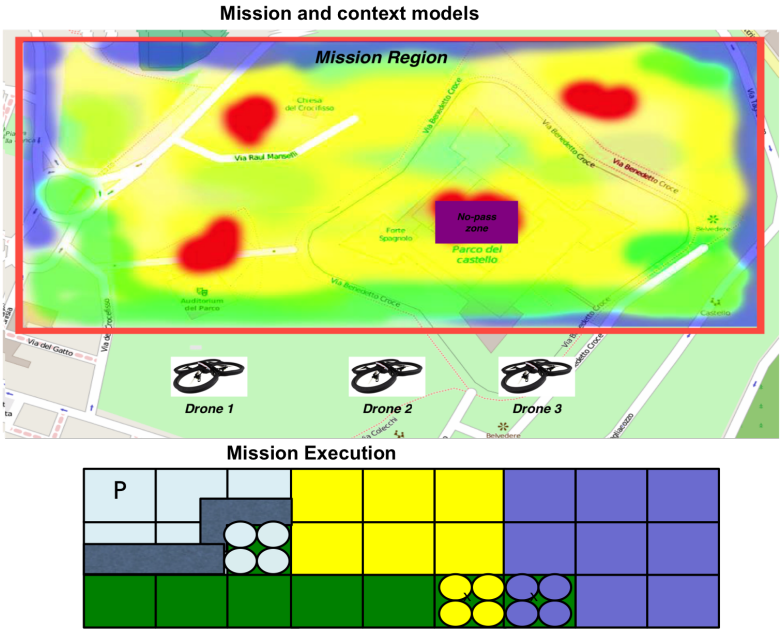


FIGURE 6.1: Motivating Scenario

specification. Our assumption is that the algorithms used by mission designers at design-time to generate behaviour plans for the agents in our framework is correct as in FLYAQ. Now, we focus on the *execution* of behaviour plans. For each agent $a_i \in A$ involved in a specific task t we generate a behaviour plan BP_i^t. We define an execution of a behaviour plan for an agent a_i as follows.

Definition 6.1 (Executing a Behaviour Plan)**.** We define an *execution of a Behaviour Plan* for an agent a_i at time τ as a tuple $EBP_\tau^i = (BP_\tau^i, S_\tau^i, R_\tau)$ where:

- BP_τ^i is the behaviour plan performed by a_i at time τ
- S_τ^i is the state of the agent a_i at time τ
- R_τ : is the return status of the Behaviour Plan R, S, F at time τ, and can be equal to either Running (R), Success (S), or Failure (F).

The behaviour state space B_i for an agent is partitioned in three partitions: *success*, *failure*, *running* when an agent a_i is executing a behavior plan BP_i (Figure 6.2). The return status of the Behaviour Plan R, S, F at time τ describes in which partition an agent's state is. The states defined in the *success* partition describe that the agent successfully completed its behavior plan. The states defined in the *running* partition describe that the agent a_i correctly executes its behavior plan at time τ. We say that an agent a_i is correctly executing a Behaviour plan BP_i at time τ if from the current state S_l the agent can continue performing its Behaviour Plan i.e., the set of conditions for

the current operation $op \in BP_i$ are satisfied. The states defined in the *failure* partition describe that the behavior plan is failing at time τ. In that point the agent should adapt i.e. change its behaviour plan, so it reaches a state in the *running* region from which it can continue executing the mission or just safely exit the mission as described in [61]. In Figure 6.2 is represented the behaviour state space B_i for an agent a_i executing a behavior plan BP_i. The *behavioural plan execution state transition* $(a \rightarrow b \rightarrow c \rightarrow d)$ represents a state transition for the agent from an initial state S_0 to a state in the success region S_7. That is only one possible state transition that could happen. There are many other possible transitions which might include the agent transitioning into a state from the failure region from where it should adapt.

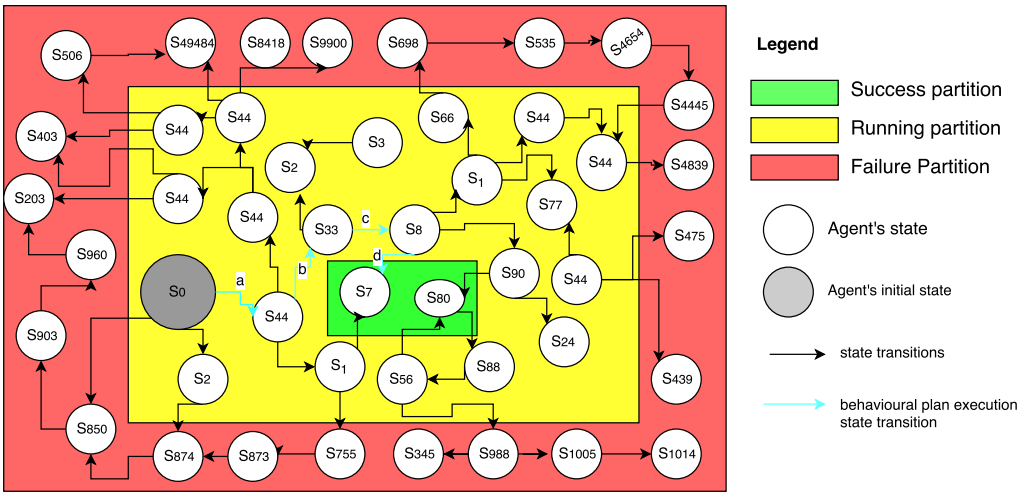


FIGURE 6.2: Partitioning behaviour state space

To model the behaviour plans of an agent a_i we will be using the Behaviour Tree Architecture because it provides a flexible mechanism for an agent to switch between different behaviour plans. Behaviour Tree (BT) is an organizational execution structure that groups behaviour units that one agent should execute as part of its mission. Each behaviour plan of an agent is modeled as a separate behavioural unit. A *Behaviour unit* is one of the basic concepts around which we define our framework. It's an executing structure for explaining what a single agent in the system should do as part of a task. A behavioural unit is modular, parametric structure that can be used across missions, projects, and organizations. We believe that modularity is important when designing, testing, and reusing complex task behaviour in robotics. Individual behaviour units allow individual behavior plans to be easily reused by other robots in other context, without the need to specify how they relate to the whole mission behavior [98]. Next, we will describe the Behaviour Tree Architecture.

6.2.2 Behaviour Tree Architecture

To model the behaviour plans of an agent a_i we will be using the Behaviour Tree Architecture because it provides a mechanism for a switching between different behaviour plans. In this section we discuss about Behaviour Tree (BT) architecture which we use for modeling the behaviour plans of the agents.

A Behaviour Tree (BT) is an organizational execution structure that groups behaviour units that one agent should execute as part of its mission. BTs were developed in the computer gaming industry, as a tool to increase modularity in the control structures of in-game opponents [98].

From [99], A Behavior Tree is a formal, tree-like graphical form that represents behaviour of individual entities which change states, make decisions, respond-to/cause events, and interact by exchanging information and/or passing control. A BT is represented as a directed tree in which the nodes are classified as root, control flow nodes, or execution nodes (leafs). For each pair of connected nodes the outgoing node is the parent and the incoming node is the child node. The root has no parents and exactly one child, the control flow nodes have one parent and at least one child, and the execution nodes have one parent and no children.

Node Type	**Symb.**	**Succeeds if**	**Fails if**	**Runs if**
Root	$\emptyset$	tree S	tree F	tree R
Selector	?	1 Ch S	N Ch F	1 Ch R
Sequence	$\longrightarrow$	N Ch S	1 Ch F	1 Ch R
Parallel	$\Longrightarrow$	$\geq S$ Ch S	$\geq F$ Ch F	otherwise
Decorator	$\diamondsuit$	varies	varies	varies
Action n	$\square$	$X_n(t) \in \mathbf{S}_n$	$X_n(t) \in \mathbf{F}_n$	$X_n(t) \in \mathbf{R}_n$
Condition n	$\bigcirc$	$X_n(t) \in \mathbf{S}_n$	$X_n(t) \in \mathbf{F}_n$	never

TABLE 6.1: The seven node types of a BT. $Ch \equiv$ children; $S \equiv$ succeeded; $F \equiv$ failed; $R \equiv$ running; $N \equiv$ children; $X_n(t) \equiv$ current state; $\mathbf{S}$, $\mathbf{F}$, $\mathbf{R} \in \mathbb{N}$ are node parameters [100]

In a BT, each node belongs to one of the seven categories listed in Table 6.1 taken from [100]: we have one root node; leaf nodes are either Actions or Conditions, while control flow nodes are Fallbacks, Sequences, Parallels, or Decorators. The execution of a BT starts from the root which sends ticks with a certain frequency to its child. A tick is an enabling signal that allows the execution of a child. When the execution of a node in the BT is allowed, it returns to the parent a status running if its execution has not finished yet, success if it has achieved its goal, or failure if is unable to continue with execution. A minimalistic example of a behaviour tree is presented on Figure 6.3. It represents the behaviour of the agent in the motivating scenario: an agent checks

if there isn't an obstacle along its trajectory. If there isn't an obstacle, it performs a movement towards p. If the agent successfully executed the movement towards point p, the behaviour tree would return success. If the agent did not reach the point p, the behaviour unit describing that behaviour plan returns failure which will be propagated to the root. In this case, an *adaptation* is needed.

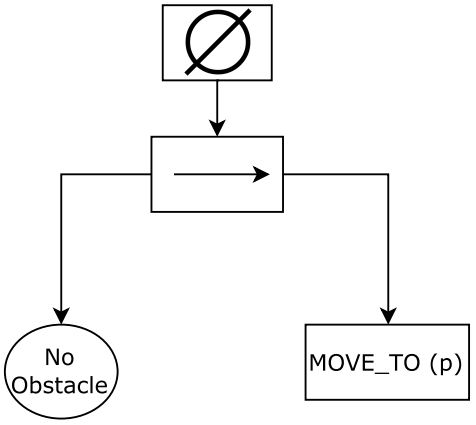


FIGURE 6.3: Behaviour plan for an agent

6.3 Framework for mission execution

In this work we propose a framework for supporting **execution** of MMRSs missions. Most of the existing works are based on the assumptions that the environment is static and that each agent has either a global communication range (can communicate with each other agent in the system) or that each agent can obtain a full knowledge of the system and the environment at any time. These assumptions, however, do not usually hold in real-world scenarios [101]. In our framework, an adaptation is performed on-the-fly every time an unexpected system or environmental feature is observed in a part of the system. That being said, a new behavioural plan is computed for one or a group of agents that are affected by it. Our framework supports on-the-fly adaptation that enables MMRSs to complete the defined mission while guaranteeing the preservation of safety constraints. As shown in Figure 6.4 the architecture of our framework consists of 3 main components:

- **Behavior Manager:** contains the behaviour model of the mission. Here, behaviour plans are stored as behaviours in a Behaviour Tree. Each agent in system is assigned a Behaviour Tree for each task. In the beginning of the mission, the Behavior Manager contains all behaviours that should be performed for completion of the mission. During mission execution, the behaviour trees may be updated as a result of violations in the behaviour plans to one or more agents.

- **Execution manager** is in charge of :(i) receiving the current behaviour model of the mission from the *Behaviour Manager*, (ii) interacting with the controllers both to send their part of mission to be executed and to receive telemetry data, (iii) checking when some conditions in the behaviour plans are violated in order to trigger the *Adaptation Manager*, and (iii) to log mission data.

- **Adaptation Manager** is a component where the adaptation happens. It receives from the *Execution Manager* the conditions that are violated and depending on the type of conditions that are violated it triggers one of its subcomponents. If safety-related conditions are violated the *Safety Manager* is always triggered. The *Safety Manager* is safety-specific adaptation component that can manage only safety-related problems. If there isn't any violation of safety-related conditions and there are mission-related conditions that are violated, the *Mission Manager* is triggered in order to perform mission problem resolution.

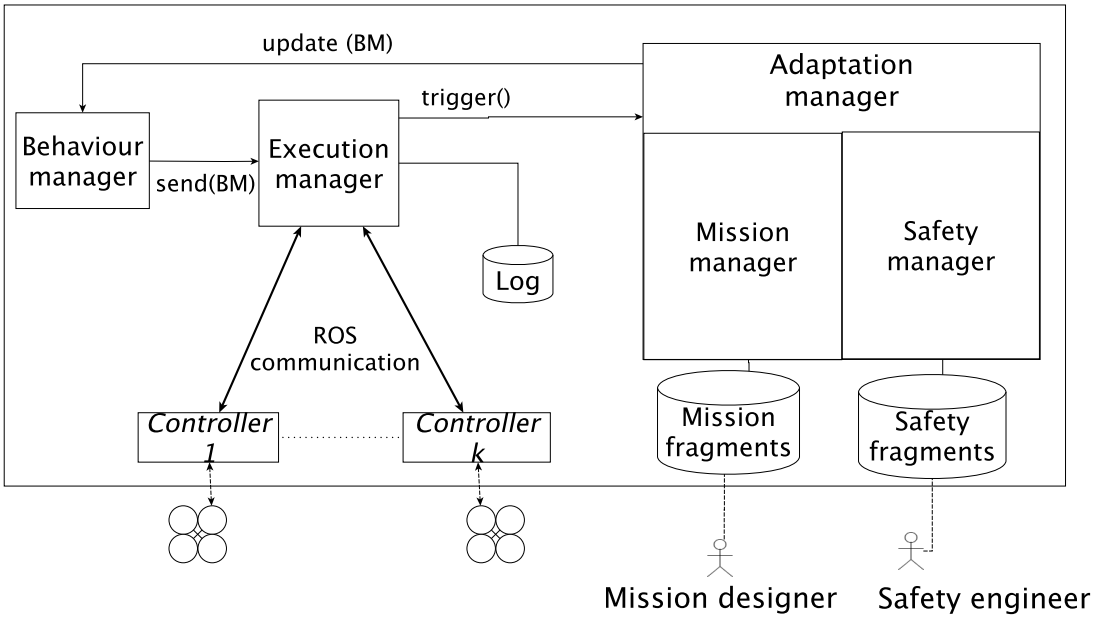


FIGURE 6.4: Overview of the execution framework

Based on the different type of issues: mission related vs. safety related the framework proposes different adaptation mechanisms which decide the right scope for an adaptation. Safety is a first class concern in our missions as robots can collaborate with humans to accomplish the mission. In this context, the system should always satisfy all safety invariants, while the mission can be partially satisfied. That is why distinguishing between safety-related and mission-related issues is of most importance in our framework. As the nature of the mission objectives is different to the safety objectives, we propose two different adaptation resolution methods: one for (potentially partial) satisfaction

of mission objectives and one for full satisfaction of safety objectives. In this work, a MMRS might need an adaptation due to the following:

- the system cannot successfully complete the defined mission (mission objective);
- agent(s) performing the current mission may physically collide (safety objective).

The *Safety Manager* contains "safety" solvers which are algorithms that generate a behavior for collision avoidance. These are agent-specific and defined independently from mission definition. The *Mission Manager* contains solvers which generate a behavior for completing parts of the mission. These are mission-specific and employed before the start of the mission.

When designing the framework we took in consideration the following types of uncertainty that the system might face and can be a reason for adaptation:

- *Changing Availability of resources:* The availability of resources for an agent can change over time (e.g., the battery level of a robot is less then a certain value, so the robot cannot finish a task);
- *Change of environment conditions:* The environment where the agents operate is dynamic (e.g., a dynamic obstacle appears, so a robot cannot finish a task).

Even though the agents participating in the mission are autonomous, they are able to dynamically form collaborative groups, called ensembles [102] to gain benefits that otherwise would not be possible. The example of such a collaborative group is an ensemble of drones that cooperate in a carbon dioxide monitoring mission represented in the motivating scenario in Figure 6.1. Multiple entities must follow certain rules in the ensemble and in return the ensemble offers certain advantages with respect to having single entities working independently. Adherence to these collective rules temporarily reduces the flexibility of collaborating entities, but has huge impact on a particular quality of the system. We can consider what happens if there is a fault on a drone and the drone can't continue with its behaviour. In this case, all the tasks that the drone did not manage to complete need to be redistributed to other entities for successful mission completion, while the faulty drone needs to adapt its behaviour plan to safely exit the mission. Another issue we can consider is a collision between drones. In that case an immediate and collective reaction by a group of drones is needed for a collision to be avoided. Here, multiple entities must adapt altogether and transactionally to perform a particular collision avoidance protocol. This demonstrates that in MMRSs two levels of adaptation are possible:

- *Isolated adaptation:* change of a single agents behavior with pre-defined behavior templates independently from the rest of the system;
- *Collective adaptation:* collective change of the behaviour of a set of agent's teamed up in an ensemble working towards a particular goal.

In this work, we mostly focus on collective adaptation, even though the framework has capabilities to perform isolated adaptation by providing a simple solution to a specific issue. The framework performs on the fly collective adaptation in a decentralized fashion in order to satisfy specific mission and safety objectives. It has mechanisms that dynamically understand which parts of the system should be selected for adaptation and can produce a solution for a specific issue.

6.3.1 Representing the MAPE-K loop structure in an entity

In this work we assume that each entity implements the MAPE-K loop. In Figure 6.5 is represented a run-time perspective of the entity's MAPE-K architecture. This perspective represents how an entity manages the execution of the mission at run-time while preserving safety constraints.

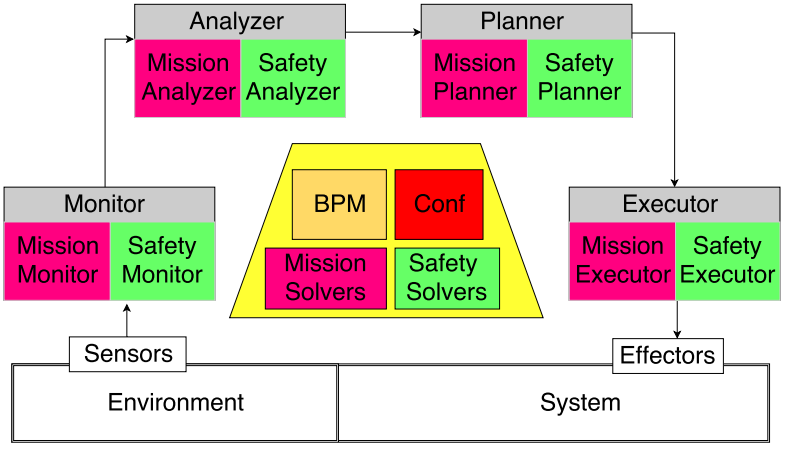


FIGURE 6.5: MAPE-K loop of an entity

In this section, we will describe each of the MAPE-K loop components for the individual entities. The MAPE-K loop comprises of 4 components operating over a Knowledge base. In order to illustrate the separation of concerns between mission-related and safety-related mechanisms for self-adaptation, there are two sub-components at each stage of the loop, one managing the mission, while the other the safety (Figure 6.5). While both sub-components are running in parallel in the Monitor and Analysis to either obtain or update information about the system or the environment, only one subcomponent is

running in the Planning and the Executor stage of the loop. In the decision of which component to run, safety has always a precedence over mission completion.

In the *Knowledge* base we define three different type of models that an entity contains. The first model is the Behaviour Tree model (BTM). The BTM contains all behaviour plans associated with the mission. Each entity has a set of behaviour plans defined at design-time, but only one behaviour plan can be *active* in one point of time during mission execution (depending on its priority). The second model is the current configuration of the entity *Conf*. This model gives information about the current resources of an entity containing information like position of the robot in the map, current level of battery, etc. The third model is a repository of the solvers it can provide i.e. *Mission Solvers* and *Safety Solvers*.

Monitoring component: This component receives stimuli from the environment and from the rest of the system (other entities in the system) and it updates the current configuration *Conf* and *BTM*. Then, it triggers the analysis component. The stimuli are values associated with specific safety-related or mission-related properties. The *Mission Monitor* keeps track of relevant mission-related information, while the *Safety Monitor* keeps track of safety-related information.

Analysis component: This component makes analysis if the *active* behaviour either *failed*, *succeeded*, or *running*. It has two sub-components Mission analyzer and Safety analyzer, both running in parallel and each making checking of the appropriate conditions (mission related vs. safety related). Depending on the analysis of the *active* behaviour it does the following:

1. Success: It references the *active* behaviour with the "next" behaviour plan in the behaviour tree model. Then, it triggers the *execution* component;
2. Failed: it triggers the *planning* component;
3. Running: it triggers the *execution* component.

Planning component: This component is triggered when the *active* behaviour returns status failed. The component starts the process of adaptation i.e. which as output generates a behaviour plan that will allow the entity to continue its mission execution or safely exit it. The planning component consists of two subcomponents: **Safety Planner** and **Mission Planner**. When the planning component is triggered, first it gets information from the knowledge about the current configuration model $Conf$. Depending on the type of configuration conditions that are violated it triggers one subcomponent or the other. Safety has precedence over mission completion, so if safety-related conditions are

violated the *Safety Planner* is always triggered. The Safety Planner is safety-specific adaptation component that manages only safety-related problems. If there isn't any violation of safety-related conditions and there are mission-related conditions that are violated, the *Mission Planer* is triggered to perform mission problem resolution.

More details about the problem resolution process will be provided in the next section where we discuss the two problem resolutions: mission problem resolution and safety problem resolution. Each of these resolutions enables two types of adaptation: (i) isolated adaptation: performed by the entity itself or (ii) collective adaptation: performed by an ensemble of entities. The behaviour plan that is generated at the end of the adaptation process is updated in the BTM and then, the execution component is triggered.

Executor component: This component receives the *active* behaviour from the Behaviour Plan Model and executes(ticks) it i.e. issues commands to the entity's effectors. When the executor component is triggered, first decides which subcomponent should be activated. Depending on which subcomponent performed the adaptation, it will activate one of the executor subcomponent correspondingly. If the safety planner was activated, the safety executor will be activated. If the mission planner was activated the mission executor will be activated. The mission executor performs mission-related behaviours, while the safety executor performs safety-related behaviours.

6.4 Formal model for Problem Resolution

In order to explain our problem resolution process we formally defined some basic concepts that are used through out the rest of the chapter. During normal conditions, each agent will perform its behaviours generated at design-time and finish its part of the mission, leading to full mission completion. However, when an agent executes a behaviour plan and reaches a state in the failure region as described in Section 6.2, the behaviour is not executing correctly. When an agent is not "executing correctly" a behaviour plan, a *problem* is triggered.

A problem is a generic structure that corresponds to different critical situations that can happen to an agent when executing a particular behaviour plan. It is generated as a result of the inadequacy between the agent configuration model and the model of its behaviour plan. It can represent situations like a state of an agent that can't cover a particular mission region because of lack of resources or a state of an agent that represents a situation of possible collision. We will discuss about the problem space we are covering in more details in the next sections. Now, we define a problem formally as follows.

Definition 6.2 (Problem)**.** A Problem is a tuple $P = (PS, f)$ where

- PS is a generic type of problems;
- $f : P- > V$ is an assignment function that assigns values for problem's properties to define the boundaries of the problem.

A Solver is a structure that receives as input an instance of a problem and produces a behaviour plan that is a solution to a particular problem. Formally it is represented as:

Definition 6.3 (Solver)**.** A Solver is a tuple $S = (P_0, PS, SS, \theta)$ where:

- P_0 is the initial problem that should be solved.
- PS is the set of all possible Problems the solver is able to solve
- SS is the set of all possible solutions
- θ is a resolution function and $(P_i, B_i) \in \theta$ represents the following:
  - $P_i \in PS$ is the problem that is addressed;
  - $B_i \in SS$ is the Behaviour Plan generated to solve the Problem (solution).

An adaptation can be performed by one or multiple agents. When multiple agents participate in the adaptation process we consider their collective behaviour. Existing approaches typically deal with multi-agent adaptive systems through isolated adaptation: each agent adapts itself independently from each other. However, here we focus on collective adaptation. This raises an important issue, i.e., identifying which parts of the system should be engaged in adaptation. This issue is not trivial at all, since a problem may be solved at different scales.

In order to explain our problem resolution process we start introducing the notions of *entity* and *ensemble*. Entities are basic building blocks in the adaptation process representing the different agents of the system (e.g., robots, ground stations, etc.). An entity can be seen as a representation of an agent that can play a *role* in the problem resolution process. Formally, we define it as follows.

Definition 6.4 (Entity)**.** An entity $y = (a_i, r)$ is defined by an agent a_i playing a role r.

A role represents the type of collaborative interaction a particular agent can participate in. Collaboration consists in managing problems and responding to problems raised. Formally is defined as follows.

Definition 6.5 (Role)**.** A Role is a tuple $R_i = (P, S)$ where:

- P is a set of problems it can produce;
- S is a set of solvers it provides.

The model of an *entity* is primarily determined by the ways it collaborates with other entities as part of an *ensemble*. In isolated adaptation the entity that triggered the problem is the same as the one that provides a solution, but in collective adaptation a solution is provided by other entities in the ensemble. An ensemble is primarily determined by the entities that collaborate to solve a particular problem. In isolated adaptation, the ensemble contains one entity, while in collective adaptation, the ensemble consists of multiple entities. In collective adaptation, the ensemble facilitates cooperation between entities by means of an information exchange at run-time. Furthermore, the collaboration between two entities is possible only if the entities can communicate between each other. Formally, we define an ensemble as follows.

Definition 6.6. An Ensemble is a dynamic run-time structure represented as a tuple $E = (A, R, \lambda)$ where:

- A is a set of agents grouped together;
- R is a set of roles the agents are playing;
- $\lambda : E \rightarrow R$ is an assignment function for which the agents are assigned their respective roles (entity definition).

Definition 6.7. A Problem Resolution is a tuple $R = (E_i, P_i, S_i)$ where E_i is the ensemble solving a problem P_i and coming with a solution S_i.

6.5 Mission Problem Resolution

Our framework implements a *Mission Planner component* as part of the Planning component in the MAPE-K loop for each entity. The mission planner receives information about the eligible mission related solvers in the Knowledge base. These are mission-specific and defined before the start of the mission. Mission-specific solvers have a set of solver constraints (configuration parameters) that reduce the space of acceptable problems (e.g., a solver for covering a geographical area might require an entity to have an active camera, enough level of battery etc. to be able to resolve a particular problem). If an entity activates an eligible solver, it generates a solution i.e. behaviour plan to complete parts of the mission. The generated solution (behaviour plan) brings the agent

into a state from which it can continue executing the mission. Here, the scope of mission specific problems is related to the nature of our definition of mission. In order to explain the *mission problem resolution process*, we frame the scope of problems for isolated and collective adaptation. An entity might perform isolated adaptation when facing with a situation where its behaviour plan trajectory needs to pass through a no-pass zone. In this case, the entity might have a solver that generates a behaviour plan for avoiding a no-pass zone. Next, we will speak in more details about collective adaptation.

6.5.1 Collective Adaptation

At design time we assigned the goal space G for a task t to a set of agents A. On Figure 6.6 the goal space G is assigned to two agents a_1 and a_2. A subregion SR_i^t of a task t that a specific agent a_i was assigned to do is decomposed in number of blocks (e.g.: the rectangle delimited by $q11$, $q16$, $q15$, and $q12$ in Figure 6.6 is one block). Then, for each block we assigned a unique identifier and associate a goal location $G_i \in Coordinates$. For each agent $a_i \in A$ involved in a specific task t we generate a behaviour plan BP_i^t covering a sub-region $SR_i^t \in G$ from its home location G_{home}^i to the last goal location G_n^i.

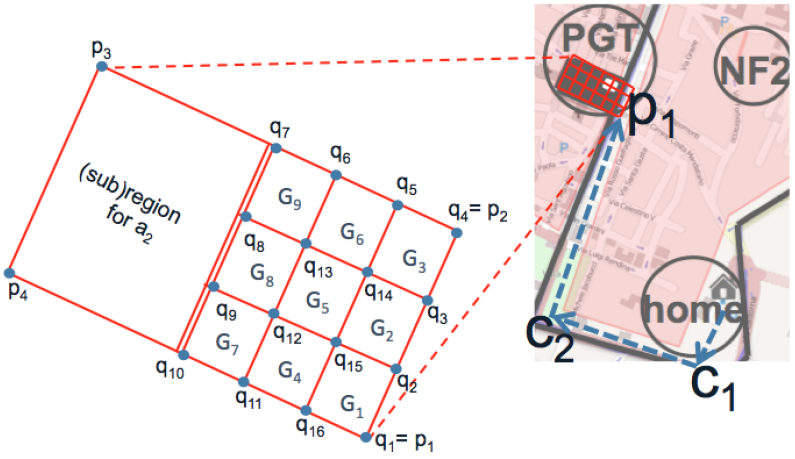


FIGURE 6.6: Task assignment for two agents

As mission-specific problems strongly depend on the type of tasks the MMRS performs, in this context, we define a mission-specific problem as a coverage path planning problem for a set of blocks in a subregion $SR_i^t \in G$ of a task t that a specific entity $y_i = (a_i, r)$ was assigned to cover, but was not able. The uncovered region is called a *problem's space*. Accordingly, we focus only on one type of mission-specific solvers which represent *strategies* for covering a region. The mission-specific solvers are formulated as algorithms solving a coverage path planning problem which depends on the type of the geometry

of the task t. Example of a solver can be an algorithm that generates a solution for covering a region with respect to a specified grid of points as in Figure 6.6.

As we work particularly with regions, a mission related problem can be decomposed on smaller problems(sub-regions). To be able to annotate the progress of task execution, we define a measure of satisfiability for a task t that gives information on how much percentage of the task goal space G_t is covered. We denote that a task t is completed if all blocks of its goal space G are covered i.e. all generated lower-level goals G_i are reached. Correspondingly, we say that a mission is *fully completed* if all tasks are completed. In contrast, a mission is partially completed if there is a task t which is not completed i.e. there is a task t with a subregion SR_i^t that contains blocks which are not covered.

In collective adaptation, the Mission Manager of an entity can decompose larger problems into smaller ones and can provide a partial solution to the initial problem. The mission manager receives information about the eligible mission related solvers in the Knowledge base and generates a solution i.e. behaviour plan. The solution is a generated behaviour plan that covers part of the problem space. Our definition of a solver in this particular context allows partial solving of a particular problem due to the fact the problems can be decomposed into smaller ones.

In this context, *Mission problem resolution* consists in reducing the problem space of a problem until the problem space is empty or until a specific time deadline is reached. We believe that cooperation in emergent application scenarios requires a new kind of problem resolution approach which is efficient in terms of short delay, so we defined a time deadline until when a solution should be found. If a full solution to the problem is found before the deadline, the mission problem resolution process don't need to wait until the deadline is reached, but it immediately returns the found solution. We formally define a solution of the mission problem resolution as follows.

Definition 6.8. A solution in the mission problem resolution R is defined as: $Sol = \max_{\emptyset \leq P_i \leq P_o} R(E_i, P_i, d)$ where E_i is the ensemble solving a problem P_i, P_0 is the initial problem that should be solved, d is a time deadline and Sol is the best solution found for that particular time deadline d.

To specify the model of the ensemble needed for problem resolution of mission related problems in details we will be using the declarative Ensemble Definition Language (EDL) [103]. The main section of the ensemble specification is the *ensemble membership* which defines the structure of the ensemble. A structure of an ensemble is defined through the ensemble roles the agents can participate in. A partial EDL Specification for the mission resolution ensembles is presented on Figure 6.7.

```
1  ensemble MissionResolution
2      id ensemble_id: Entity
3      membership
4          roles
5              Leader : Entity
6              Solver_Agent [1..n]: Entity
7      constraints
8          constraint Solver_Agent.hascommpath(Leader)==⊤
9      fitness sum Leader.solution.quality+Solver_Agent.solution.quality
10     knowledge exchange
11         Solver_Agent.target = Leader.r_id
12         Solver_Agent.Problem = Leader.Problem
13         Leader.solution=Solver_Agent.solution
```

FIGURE 6.7: EDL Specification for mission resolution ensembles

To identify the ensemble, we declare the id of the entity playing the role Leader to be the id of the ensemble essentially saying that instances of this ensemble type cannot be created without being associated with a unique entity instance, which can be seen as a sort of coordinator of the ensemble.

The ensemble membership function consists of three sections. First, the structure of the ensemble is defined by declaring the ensemble roles that the entities can play. In our case, an agent can play one of the following roles in the mission resolution ensemble:

- Leader: an entity that triggers a problem P_k and leads the ensemble formation;
- Solver_Agent: an entity that participate in the solution creation of the aforementioned problem P_k.

An agent can play more then one role in the ensemble i.e. it can be a leader and a solver_agent (it can trigger a problem, but at a same time it can provide a partial solution to the problem it triggered).

Next, we place semantic constraints, represented by the constraint expression. In our scenario a constraint for an entity to be part of the ensemble is that there must be a *communication path* between the corresponding entity and its leader. What we mean by communication path is that if the ensemble leader sends a message in the environment, the information can be transferred from one entity to another, and all entities that are able to receive the information are in the communication path of the leader. In other words, each entity in the ensemble should have a neighborhood region that is overlapping with the neighborhood region of at least one other entity from the same ensemble. Neighborhood NR is a region of an entity e_i that gives information with which other entities can communicate in a particular point of time τ. At any point of time during mission execution, each entity in the system has partial view of the system that consists of a list of entities *neighbours* that can communicate with. The third part of the membership definition is the fitness function, specified with a numeric expression. The fitness function provides information about which aspect of the ensemble membership

should be optimized. That gives the framework a way to decide which entities should participate in the ensemble formation. More precisely, ensemble instances will be created in such a way to maximize the fitness function. In our example, the fitness function is calculated as a sum of the solution quality provided by all entities in the communication range of the leader. Finally, knowledge exchange is specified, creating an information exchange between the members of the ensemble. In our case, we have exchange of three types of information. First, is the information on which entity is the leader in the ensemble (line 11), second each of the entities in the ensemble has information about the problem that should be solved (line 12) and last is the solution that the leader obtains for each entity in the ensemble (line 13).

6.5.2 Mission Problem Resolution Algorithm

We propose a best-effort approach for mission problem resolution which is efficient in terms of short delay and which does not require knowledge of which and how many entities(agents) are in the system in a particular point of time. To realize our approach, we abstractly define a recursive best-effort algorithm that covers the procedure for mission problem resolution. The algorithm starts from an entity e_i that originally detected a Problem P_i and expects to commit a solution until a time deadline d. Further recursive calls are propagated to other entities in the environment using events.

The algorithm consists of three phases:

1. discover phase
2. construct phase
3. commit phase

In the **discover phase**, the possible entities that can participate in the mission problem resolution are found. Each of the entities that can contribute with a solution to a specific problem and have a communication path towards the leader are discovered and links between each of the corresponding entities is created. At any point of time during mission execution, each entity in the system has partial view of the system that consists of a list of entities *neighbours* that can communicate with. Neighborhood NR is a region of an entity e_i that gives information with which other entities can communicate in a particular point of time τ. Each entity that contains an active solver(solver for which the preconditions for activation are fulfilled) can contribute towards a solution to the mission-specific problem if it is in a *communication range* with the entity that triggered the problem. In the **discover phase**, those entities are found and a *temporary* ensemble

is formed. The *temporary* ensemble consists of all possible entities that might participate in the resolution process.

In the **construct phase**, each discovered entity in the *temporary* ensemble contributes towards the global solution formation. Each of the entities in the communication range contributes towards the global solution by resolving a particular sub-problem of the general problem, and produces a solution with some specific quality q until a specific deadline d is reached. Hence, solutions are composed from multiple solvers from different entities, the same way problems can be decomposed on multiple sub-problems. In the end, the entity that triggered the problem resolution process decides which is the *best* solution and which part of the temporary ensemble contributes towards it.

In the **commit phase**, the leader of the ensemble knows how well each of the entities can solve a particular sub-problem so it sends request to the entities that can contribute with the best solution to commit their resources. Meanwhile, some of the entities might leave the *temporary* ensemble due to a lack of resources, because they have commitment for another problem resolution process or because they are not anymore in the neighborhood of the entity that triggered the problem. After entities commit their resources for execution, a *stable ensemble* is formed and the *final solution* is obtained. The ensemble that provided the final solution is called *stable ensemble*. It is the ensemble that provides guarantees about the proposed solution i.e. if all behaviours from all entities in the final ensemble execute correctly (according to the generated behaviour plan), the final solution will be guaranteed.

For our algorithm to work we take in consideration the following assumptions:

- All missions problems can be decomposed on sub-problems
- The entity that triggered the problem does not fail between the time it triggers a problem and commits a solution to an ensemble
- There exists connectivity between the entity that triggered the problem and at least one other entity in the environment for an adaptation to happen (that entity can be the same entity that triggered the problem)
- There is not a noticeable difference in the resource configuration for one entity between the time it proposes a solution and commits its solution
- There is not a noticeable difference in the connectivity formation between entities that proposed solution and commit their solutions

- The last two assumptions are translated to the following assumption: there isn't a noticeable difference between the solution quality the moment a solution is proposed and the moment a solution is committed.

The decentralized mission manager for an entity $r_i \in R$ is shown in Figure 6.8 in the form of a state machine, which is executed by each entity in the system. It is presented in the form of pseudo-code that closely represents the syntax of the P programming language. A P program comprises of concurrently executing state machines communicating asynchronously with each other using events accompanied by typed data values. Each state machine has an input queue and machine-local store for a collection of variables. Each state has a set of event-handlers which get executed on receiving the corresponding event. The function $send(r_k, ev, d)$ is used to send an event ev with payload data d to target machine r_k. An entity r_i broadcasts event ev with payload d to all the robots in its communication range using the function $broadcast(ev, d)$ (more details about P language are available at [104]).

Figure 6.8 shows the algorithm that encodes the mission-problem resolution state machine. This state machine has three states: *Discover*, *ConstructSolution* and *RequestCommit*. It contains the following variables which are important for understanding the code: r_i - represents the id of the robot that executes the state machine, P is the whole problem space for which the entity has already proposed a solution, S is the global solution that the leader obtains, $timerV$ is a state machine that is instantiated when a problem is triggered, sol is a local solution provided by an entity in the ensemble.
Now, we will go through the code. The algorithm includes the following important steps:

Lines 2-8. The mission-resolution manager starts executing in the *Discover* state. When a mission related problem is triggered, the solver of the entity r_i is invoked and a solution *sol* is calculated for the specific problem *Pi*. Function *callSolvers* (**line 4**) is beyond the scope of this paper but may generally exploit various mission-specific solvers and provide corresponding full or partial solutions. After the solution is calculated, the machine creates an instance of a Timer machine, starts the timer and goes to *ConstructSolution* state.

Lines 24-43. Upon entering the *ConstructSolution* state, it checks if the solution can fully or partially solve the triggered problem P_i. If we have full solution to the problem P_i, the state machine transits to state *RequestCommit* state(**line 32**). If there is a partial solution, the problem space is reduced to P_j (**line 30**) and the agent broadcasts problem P_j in its neighborhood (communicates the problem P_j with all entities in its communication range).

```
machine MissionResolution–
  start state Discover–
    on TriggerMissionProblem(P_i) do–
      sol = callSolvers(P_i);
      timerV = new Timer(this);
      StartTimer(timerV);
      goto ConstructSolution;
    ″
    on ReqForSolver(P_j, r_j, d) do–
      if (P_j ∉ P) then
        sol = callSolver(P_j);
        if (sol ≠ ∅ ∧ t ≤ d) then
            P = P ∪ P_j;
            send(r_j, solutionfound, sol);
        end if
      end if
    ″
    on Commit(sol, r_j)–
      sol=check(sol);
      update(sol);
      send(r_j, confirm, (r_j, sol))
    ″
  ″
  state ConstructSolution–
    defer ReqForSolver, Commit;
    entry –
      if (sol ≠ ∅)
        Target = r_i;
        S = sol;
        P_j = P_i \ p(sol)
        if (P_j == ∅) then
          goto RequestCommit;
        end if
      else
        broadcast (ReqForSolver, (P_j, r_i));
      end if
    ″
    on SolutionFound(r_j, sol) do–
      Target = Target ∪ r_j
      S = S ∪ sol
    ″
    on TIMEOUT push RequestCommit;
  ″
  state RequestCommit–
    entry–
      < Sbest, Target > = find˙best˙solution(S,Target)
      S = ∅;
      Rrecv = ∅;
      foreach t ∈ Target
        send(t, commit, (Sbest(t), r_i))
      end
    ″
    on Confirm(r_j, solution)–
      S = S ∪ solution;
      R_recv = R_recv ∪ r_j;
      if (sizeof(R_recv) = —Target—) then
        update(sol);
        goto Discover;
      end if
    ″
  ″
″
```

FIGURE 6.8: Mission problem resolution algorithm

The events *ReqForSolver* and *SolutionFound* are used for ensemble formation. They create the links between the different ensemble participants that provide solution in the

resolution process. For each link, a corresponding problem communication is derived. For each problem communication, a combination of potential solutions is identified across all reachable entities and returned. The entity that triggered the Problem stays in the *ConstructSolution* state until the timeout is reached. When the *Timer* machine sends *TIMEOUT* event to the entity that triggered the problem, the same entity goes to *RequestCommit* state.

Lines 44-62. Upon entering the *RequestCommit* state, the function *find_best_solution* is executed to identify the best solution *sbest* and which combination of entities contributes to *sbest* (**line 46**). The function *find_best_solution* is beyond the scope of this paper but generally is a domain and application specific. Finally, the leader sends the event *commit* to ask the ensemble participants in the specific problem resolution to commit their resources. It should be noted that in the time between the solution is proposed and the solution is chosen, some deviations of resources important for problem resolution might be encountered. That is why the check function (**line 19**) checks the change in the proposed and the current solution, it updates the Behaviour Tree in the *update()* function (**line 20**) with the current state of the local solution and sends confirmation to the leader. When the leader receives confirmation from all ensemble participants (**line 53**) it updates its Behaviour Tree and transits to *Discover* State. The algorithm provided in Figure 6.8 is able to resolve only one problem resolution triggered by one entity at one point of time without any recursion. Here, all the members in the ensemble can directly communicate with the leader. We propose an extension of the algorithm for resolving multiple problems triggered by different entities at a same time. Furthermore, sub-problems are recursively triggered across different entities in the system in order to have a larger range of possible solutions i.e. a leader can obtain a solution from an entity that can't directly communicate with it, but through another entity in the ensemble that is in its communication range. We define a data type $MissionResolution = (PI, Ei, Si, ES, deadline, parent, PO, FE, FS)$ which is a tuple that contains information about one problem resolution (one ensemble) in which the entity participates. It contains the following variables:

- *P0* is the initial problem that is triggered by the entity. It can be sent by another entity that requests collaborators for resolution of a larger problem or it can be generated as a result of the inadequacy between the entity configuration model and the model of its behaviour plan.
- *parent* gives information from where the initial problem originates. If it is generated by the same entity then parent receives the id of the entity r_i.

- *PI* is a reduced version of the problem that is obtained after the entity proposes some solution. We use the value of PI for identifying the different resolution processes in which the entity participates.
- *Ei* stores the information about which are the entities that participate in the temporary ensemble formation,
- *Si* is the solution proposed by the temporary ensemble
- *ES* is a matrix which gives information which entity proposed which solution
- *deadline* is a timer machine that is instantiated when a problem is triggered and decides until which period of time an ensemble formation is allowed.
- *FE* is the stable ensemble which is obtained after commitment.
- *FS* is the final solution proposed by the stable ensemble

We can represent each instance of the resolution process as a tree, which we will call problem resolution tree (Figure 6.9). On top of the tree is an entity that triggered the general problem $P0$, while each node in the lower levels in the tree represents an entity that decompose the problem of its father entity and provides a partial/full solution. In the end, we have a resolution tree consisting of nodes representing the entities in one possible instance of the problem resolution process. We define a *hierarchical order* in the resolution tree depending on the *communication range* of the entities. The order of an entity in the tree is defined through the hop counter that refers to the number of intermediate entities through which an information must pass between the entity and the leader in the ensemble. For example in Figure 6.9, the leader e1 might not be able to communicate its problem P1 with the entities e5,..., e10 which are the leafs in the resolution tree because of the limitation in its communication range, but it might need few entities from the ensemble that are able to transit the information to the leafs (like entities e2, e3, e4) which are in the communication range of the leader e1, but also in the communication range of the leafs in the tree. The order of the entities that can directly communicate with the leader is higher comparing to the entities that need an intermediate entity to relay (in Figure 6.9, the leader e1 has the highest order, while the leafs e5,...e10 have the lowest order).

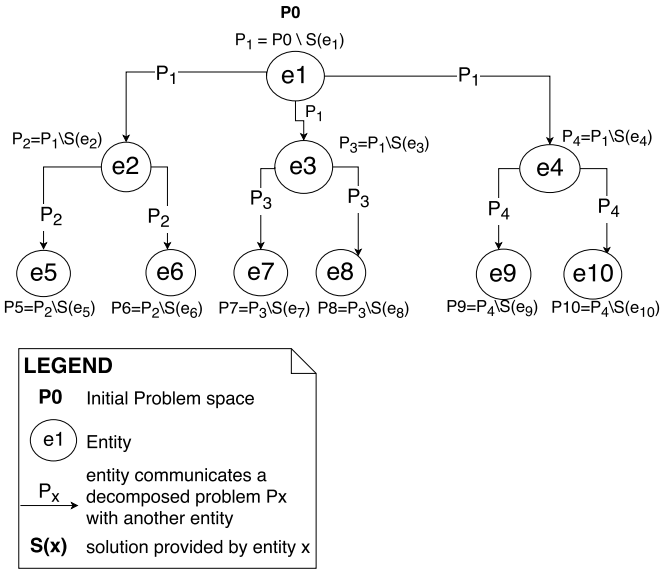


FIGURE 6.9: Problem Resolution Tree

A formal definition of the problem resolution tree is as follows.

Definition 6.9 (Problem Resolution Tree)**.** A problem resolution tree is a tuple $T = (root, Ei, L)$ where:

- *root* is the entity that triggered the top-level problem $P0$
- E_i are the nodes in the tree represented through the entities that decompose and solve part of the top-level problem
- $L : N \mapsto N$are parent-child links between entities that are able to communicate between each other. It is a function that represents problems/solutions communications from the root to its children.

Each child in the tree decompose the problem received from its parent, so in the end we have a resolution tree where the leaves are entities that contain the smallest subset of the problem space. Each problem resolution tree represents only one possible instantiation of the problem resolution process. When the problem resolution tree have a full solution, the leafs' problem space is an empty set.

We define a quality q of a solution S_i proposed by an entity in the problem resolution tree based on two factors: (i) *closeness* to the entity that triggered the problem, (ii) intrinsic quality given by the entity. What we mean by closeness to the entity is the following: in one instance of the problem resolution tree, if there is an entity e_k that has a *higher-order in the hierarchy* in the problem resolution tree and can propose a solution s_k to a sub-problem p_k, then we consider that the solution s_k has precedence over solutions that

are able to solve the same sub-problem p_k, but are coming from other entities that have lower order in the hierarchy in this instance of the problem resolution process. Because the communication between entities is limited, the algorithm is searching for solutions closer to the entity that triggered the problem p_k and if it finds one, it stops the search for other solutions that are generated from entities that might produce solutions with better intrinsic quality, but are more distant from the leader in the problem resolution tree. Thus, when we speak about hierarchy, we consider hierarchy of entities in terms of the problem resolution tree: the nodes that are closer to the root (meaning they need less number of hops to communicate with the root) have a higher order in the hierarchy comparing to nodes that are lower in the branching. Root has the highest order in the hierarchy, while the leafs have the lowest order in the resolution tree.

6.5.3 Correctness and completeness

To resolve mission resolution problems we used a gossiping protocol that aims to disseminate the information about a specific problem and finds a solution. To prove the correctness and completeness of the approach, we need to prove correctness and completeness of the algorithm.

6.5.3.1 Correctness

In order to prove that our algorithm is correct, we assume that the solvers provided by the different entities are correct. Correctness of a solver means that an entity's solver can generate a behaviour (solution) to resolve a particular mission related problem with a quality q_a. If that behaviour is executed correctly we have a correct adaptation.

The leader when calculating the best solution for a particular problem, doesn't have the exact information on how each entity in the ensemble contributes towards the final solution. The only information the leader of the problem resolution process has when making the decision is how each entity that is directly reachable (it can directly communicates with) can help in resolving part of the bigger problem of the leader. All reachable entities might have formed sub-ensembles that contribute towards the final solution, but the leader does not have that information. For example a leader might be able to communicate with two entities that can provide some solution to the initial problem. Each of those two entities might have formed a temporary sub-ensemble. The two temporary sub-ensembles are on the same hierarchical level and they might have one entity in-common, however they belong to different instances of the resolution process and can be represented with two different problem resolution trees. When the leader decides for the final solution it might consider a combination of both sub-ensembles which

create the final solution and chooses a final *stable ensemble* that is a combination of both sub-ensembles. In each level of the problem resolution tree each node has calculated the "best solution" that can be provided by its leafs. The process is repeating and in the end, the root of the tree calculates the "best solution". The best solution provided by the stable ensemble can be represented in one instance of the problem resolution tree as in Figure 6.9.

That being said, the assumption is that there should not be a noticeable difference in the solution quality between the moment a solution is proposed and a solution is chosen. Solution quality will remain the same if there isn't any change in the *entity's resources* and in the *connectivity formation* of the ensemble. That is why we assume that the time between a solution is proposed and a solution is committed is within seconds, so that there isn't any change in the *connectivity formation* of the ensemble (**Assumption 1**). However, change in the resources for the proposed solution might come if an entity participate in two different resolution processes triggered by different leaders. During problem resolution, entities might propose solutions for different problems in different ensembles. As the resources of the entities are limited, we made the assumption that the entity will not participate in two different resolution processes (resolution processes triggered by two different leaders) which overlap in the usage of the resources i.e. if an entity participates in two different resolution processes from different leaders the usage of resources will not overlap (**Assumption 2**). However, there is another case that impacts the quality of the solution and that is when an entity tries to propose multiple solutions in one problem resolution process. Here, we will proof that with our algorithm that won't be possible. Thus, we define correctness as follows.

Theorem 6.10 (Correctness)**.** *If an entity e triggers a problem P_i and Assumption 1 and Assumption 2 are true, then the Mission Problem Resolution Algorithm finds and computes the best quality solution S_i proposed by an Ensemble E that is in the communication range of the entity e.*

Proof. As we mentioned earlier we assume that all mission related problems can be decomposed. Let's say we have an entity e that triggered the problem Pi. Pi can be decomposed on m different different ways. For each different decomposition there is a sequence of local solutions proposed by an ensemble E_m that combined together give a global solution S_m. The global solution S_m is a sequence of local solutions $s_0, s_1, ..., s_k$, each of them with a particular quality $q_0, q_q, ..., q_k$ correspondingly. Let's assume there exists an ensemble consisting of n entities for which there is a communication path between them and the leader (the ensemble might include the leader) and that they can provide the best final solution S_i to a problem P_i. Correspondingly, we can decompose the solution to a sequence of local solutions $S_i = (s_0, s_1, ..., s_n)$ each with a particular

quality $q_0, q_1, ..., q_n$. We can represent that ensemble using the problem resolution tree (Figure 6.9). In order to prove that the algorithm is correct, we need to prove that the computed solution S_i by the leader of the stable ensemble E_n is the best for the problem P_i. To prove that, we use the problem resolution tree of the *stable ensemble*. The root of the tree is the leader. We need to prove that the solution calculated by the leader has the highest quality i.e. $S_i = (s_0, s_1, ..., s_n)$ each with a particular quality $q_0, q_1, ..., q_n$. As our mission problem resolution is recursive at each node in the resolution tree, the algorithm calculates the best solution by considering the best combination of solutions proposed by its children. After calculating the best solution, it sends that solution to its parent. Starting from the leafs of the tree, the nodes calculate the best combination of solutions. In the end, the leader composes all combinations and calculates the best combination of solutions proposed by its children. If an entity proposed solutions to multiple problems in the same instance of the problem resolution tree, the algorithm will return a value which might not be correct because of lack of resources for the entities that proposed multiple solutions.

That is why we need to have (i) a set of n *different agents* in the *stable ensemble* contributing towards the final solution S_i as a precondition for the leader to be able to correctly calculate the best possible solution. Our algorithm should not allow for one entity to participate with multiple different solutions in a same ensemble because as we mentioned earlier it might not be possible for one entity to perform multiple solutions due to a lack of resources.

To prove (i), we need to to prove that there is no possibility for a *communication loop* in one instance of the problem resolution process (problem resolution tree). What is considered as a possible loop in this distributed algorithm is a situation where one entity communicates a sub-problem $P_j \in P_i$ with another entity that reduce the problem to $P_k \in P_i$ and communicates that problem to the first entity that triggered P_i. We can imagine a situation where before an entity commits its resources to a particular ensemble, it might propose solutions for other ensembles to resolve different problems, so in that case we might encounter a situation where the first entity will propose a solution to a sub-problem that was not able to resolve it before taking in consideration the whole nature of the problem (ex. in a previous iteration the entity proposed a solution to a more general problem and if it commits its resources to that solution, it might not be able to resolve the smaller problem that requires a solution in this iteration). To avoid that, each entity that runs the algorithm checks if the problem that is being received for resolution is some type of sub-problem of a more general problem that was being resolved in a previous iteration. If that is the case, the problem resolution procedure will not start i.e. the entity will not participate in the execution of the sub-problem.

In order to prove the correctness of the algorithm, we verified (using model-checking) the following property: (i) for each problem resolution tree, all nodes in the tree represent different entities in the system. We used Zing model-checker [105] to systematically test our algorithm represented in the state-machine based programming language P. From here we can conclude that for each instance of the resolution procedure for a problem P_i we have a set of n *different entities* that contribute towards the solution S_i. In other words, there aren't two nodes in the tree that have reference towards same entity.

Hence, we proved correctness of our algorithm.

□

6.5.3.2 Completeness

Completeness of the approach is defined based on assumptions about the connectivity between the agents and the stability of resources and connectivity. First, we assume that we have complete connectivity between agents meaning that starting from each agent we can broadcast a message that will reach all agents in the environment for a specific deadline d. Second, we assume that we have stable connectivity between agents which means that the connection between the agents will not disrupt during the adaptation process. Third, we assume that we have stable resources during adaptation which means that there isnt a change in the resources important for the agent to execute the solution it proposes. Taking in consideration these assumptions, we define the completeness of the approach as follows:

Theorem 6.11 (Completeness)**.** *If a solution S_i for a specific problem P_i exists and we have a deadline d to find it, the algorithm is able to find it.*

Proof. If we represent one instance of the resolution process of problem P_i as function that reduces problem P_i, then we can represent each reduction of P_i in a monotonic decreasing sequence where each instance represents the appropriate reductions of the problem P_i. Obviously, if a sequence is decreasing and is bounded below by a minimum, in some finite time we will reach that minimum. In our case, the minimum would be the problem that corresponds to the final solution S_i, which in ideal situation will be the empty set. Having that for all instances of the mission resolution process, we can conclude completeness. □

6.6 Safety Problem Resolution

In this section, we discuss about the adaptation resolution problem related to safety problems. To be able to model specific safety related behaviours, we discuss few properties of agents and how they are connected to safety.

Here, what we mean by safety is the capability of avoiding undesired outcomes. In the MMMRs domain we consider actions that has catastrophic outcomes for safety and can't be undone by a finite sequence of other actions.

We define what a *safe MMRS* is. Informally, a MMRS is safe if:

- the configurations of all agents are safe. We denote with $conf(a_i, \tau)$ a boolean function that gives information if the configuration of the agent a_i is safe at a particular time τ.
- the mutual interdependence of all robots configuration pairs involved during the whole mission is safe. We denote with $Dconf(a_i, a_j, \tau)$ a boolean function that represents the dependency of the configurations of agents a_i and a_j at time τ. If it is *safe*, the function returns true, otherwise if it is *unsafe*, it returns false.

In the following we formally specify what a safe MMRS is. With MMRS we denote the set of all agents performing the mission (Definition 4.7) and with T the total mission execution time.

Definition 6.12 (Safety)**.** We say a MMRS system is **safe** at time $\tau \in T$ if and only if the following two points are valid:

- $\forall a_i \in MMRS; conf(a_i, \tau) = true.$
- $\forall a_i, a_j \in MMRS; Dconf(a_i, a_j, \tau) = true.$

Before defining what types of configurations of an agent are safe (i.e. when $conf(a_i, \tau) = true$), and what type of interdependence of configurations of two robots is safe (i.e. $Dconf(a_i, a_j, \tau) = true$) we discuss few properties of agents.

We take a snapshot of the mission at a particular time $\tau \in T$. Now we define the following. We denote with OBS_τ the region of all obstacles (known and uknown) in the environment at time τ. We mentioned earlier that we define an Obstacle $o \in OBS_\tau$ as a region in the environment that should not intersect with the region of operation of an active robot(agent) to not jeopardize safety. We denote by $VR^i_\tau \in R$ the "visible region" in which an agent a_i can "observe" its local environment (obstacles and other agent's

location) at time τ and by SR^i_τ the **safety region** of an agent a_i at time τ that represents a region that is the absolute minimum separation for safety that must be maintained during a close encounter with other (robots) agents or with a static/dynamic obstacle . In this work, we focus on one representation of safety defined through the concept of *collision*. We define *collision* as a situation when the safety zone of a robot is overlapping a region of an object or a safety zone of another robot at time τ. In that context, we say that a MMR system is **safe**, if no collision happens during mission execution. Taking that in consideration, we define $conf(a_i, \tau)$ and $Dconf(a_i, a_j, \tau)$ in relation to collision avoidance. They are represented as safety invariants that must always be satisfied during mission execution i.e. their value must always be *true*.

Definition 6.13 (Safety Invariants)**.** We define the following two safey invariants: $\forall i, j \in MMRS \quad and \quad \forall \tau \in T$

1. $Dconf(a_i, \tau) = true \iff SR^i_\tau \cap SR^j_\tau = \emptyset$
2. $conf(a_i, \tau) = true \iff \forall o \in OBS_\tau; SR^i_\tau \cap o = \emptyset$

We defined SAFETY in Definition 6.13. with 1) and 2) as absence of collisions. 1) states that there will be no collision between agents, while 2) states that there will be no collision between agents and obstacles during mission execution.

Our framework implements a safety planner as part of the Planning component in the MAPE-K loop for each entity. The knowledge base of the MAPE-K loop contains a catalogue of correct obstacle avoidance algorithms as solvers which can be activated and able to provide a solution for an agent in a specific situation. These are solvers that can be reused in different application scenarios and missions independently of the type of the domain. Here, the scope of safety specific problems is independent of the nature of the definition of the mission.

In order to explain the safety problem resolution process, we frame the scope of safety problems for isolated and collective adaptation. In our work, we envision resolution of the following types of safety problems: (i) collision with a static obstacle; (ii) collision with a dynamic obstacle; (iii) collision between agents. In the case of static/dynamic obstacle avoidance, an entity performs isolated adaptation i.e. one entity generates a solution(behaviour plan) to the problem. The generated solution (behaviour plan) brings the entity into a state from which it can continue executing the mission. Next, we will speak in more details about collective adaptation.

6.6.1 Collective adaptation

For the third type of problems (collision avoidance between agents), our framework provides a palette of coordination protocols for the agents to be able to perform collision avoidance maneuvers. To resolve safety problems related to collision between agents, our framework uses the concept of ensembles described above. Agents dynamically form collaborative groups using attribute-based communication ensembles as described in [106] to gain benefits that otherwise would not be possible. In safety problem resolution, the agents must follow certain rules and in return the ensemble offers a guarantee that if all single agents follow the rules, safety will be preserved. Adherence to these collective rules temporarily reduces the flexibility of the collaborating agents, but has a strongly positive impact on safety. Comparing to mission resolution where those collective rules are more flexible, the safety resolution requires stronger, more precise, and detailed rules.

A safety ensemble is primary determined by the agents that collaborate to solve a safety solver. In a collision between multiple agents, our safety resolution procedure consists of: (i) a protocol for on-the-fly ensemble formation for safety resolution and (ii) a recursive function that is initially called locally by the ensemble leader to select and commit a solution. In the safety problem resolution, all entities in the ensemble must participate in the solution creation because full solution is required. What we mean here is that we treat safety as binary (MMRS is safe or not). In contrast, in the mission resolution a partial solution is enough for solving a particular problem. If one agent fails to comply to the rules in the ensemble, safety will be compromised. Here, the shape and structure of the ensemble is strongly correlated with the type of the safety problem due to the fact that all involved participants need to generate their corresponding behaviours to guarantee the safety of the system.

```
1  ensemble SafetyResolution
2      id ensemble_id: Entity
3      membership
4          roles
5              Leader : Entity
6              Solver_Agent [1..n]: Entity
7      constraints
8          constraint Solver_Agent.hascommpath(Leader)==⊤
9          constraint ∃ Solver_Agent.solver
10         compatible(Solver_Agent.solver, Leader.solver)==⊤
11     fitness Leader.solution.quality
12     knowledge exchange
13         Leader.Problem=Solver_Agent.Problem(conflict_r)
14         Solver_Agent.solver = Leader.solver(A, attr)
15         Leader.solution=Solver_Agent.solution
```

FIGURE 6.10: EDL Specification for safety resolution ensembles

We specify the ensemble type used for safety problem resolution on Figure 6.10. To identify the ensemble, we declare a leader agent to be the id of the ensemble - essentially saying that instances of this ensemble type cannot be created without the ensemble being associated with a unique entity instance, which can be seen as a coordinator of the ensemble. A Leader of an ensemble for safety resolution is an agent that leads the ensemble formation and decides for a safety resolution protocol. Example could be a fixed camera positioned in a particular point in the environment that checks if there is a possibility for collision between two or more robots or a robot that notices another robot in its visible region. Unlike the leader in the mission resolution ensemble, the leader in a safety resolution has knowledge of all the possible ensemble participants when decides for a solution type and when it starts the coordination of the problem resolution process.

As we can see from Figure 6.10, the ensemble membership function consists of three sections.

First, the structure of the ensemble is defined by declaring the ensemble roles the agents can play. In our case, same as in the mission resolution, an agent can play one of the following roles in the safety resolution ensemble:

- Leader - an agent that has the highest safety index and leads the ensemble formation;
- Solver_Agent - an agent that can provide partial solution that contributes towards the final solution.

Second, we place semantic constraints, represented by the constraint expression. In our safety resolution ensemble a constraint for an agent to be part of the ensemble is that there must be a communication link between the corresponding agent and its leader. What we mean here is that the leader can communicate with all ensemble participants. The other very important aspect in the ensemble formation is the solution space. In contrast to the mission resolution ensemble, a safety resolution ensemble must provide a full solution, so we put that as a constraint. Full solution consists of combination of behaviours generated by all participants in the ensemble. What we mean is that all ensemble participants agree to follow a specific protocol suggested by the leader i.e. each entity in the ensemble must have a solver compatible to the solver proposed by the leader. Our assumption is that each entity has at least one solver that is compatible to the solvers of the rest of the system. We consider this assumption reasonable because we consider safety independently from the mission, so all safety solvers can be independently embedded in the knowledge base before the start of the mission independently of their type.

The third part of the membership definition is the fitness function, providing information about which aspect of the ensemble membership should be optimized. In our example, the fitness function is represented as maximized solution quality of the leader that coordinates all ensemble participants.

Finally, knowledge exchange is specified, creating an information exchange between the members of the ensemble. In our case, we have exchange of three types of information. First, it is the information of the agents' conflict region $conflict\_r$, which is calculated by the position of the agent and its corresponding speed (line 13). $conflict\_r$ is the safety region of an agent in a specific time interval during the mission execution. Second, each of the entities in the ensemble receives an information about a specific solver proposed by the Leader. Each entity in the ensemble receives information about a collision avoidance algorithm A and the attributes of the algorithm $attr$ (example of attributes of the algorithm might be the central point around which the entities will perform the collision avoidance protocol, their corresponding speed, etc.) (line 14). Third, the leader gets information about the solution of each entity in the ensemble (line 15).

We defined the following protocol (Figure 6.11) that is used for on-the-fly ensemble formation when agents discover that they are facing with a possible collision among them. Each agent starts executing in the *Discover* state. If an agent a_i notices other agents in its visible region, it broadcasts the $ReqforSafetyRegion$ event with identifier for the VR - visible region, the time τ when the message is sent and the identifier of the agent id, asking for the safety region SR of the robots in its "visible region" during some time period $\Delta\tau$. Then it goes to the *WaitForResponse* state.

Depending of state it is, when an agent a_i receives the event *ReqforSafetyRegion(msg)*, it generates a message $m = (a_i : int; conflict\_region : R; ensemble\_state : int)$, where a_i is the identifier of the agent creating the message, $conflict\_region$ is a region in the environment that should not intersect with the region of operation of an active agent. If the agent is not part of an ensemble, the conflict region is equal to the agent's $SR^i_\Delta\tau$, while if the agent is part of an ensemble, it represents the *execution region of the ensemble*, which is the union of the $SR^i_\Delta\tau$ of all ensemble constituents. The $ensemble\_state$ gives information if the agent is part of an ensemble and if it is, it gives information in which phase of operation the ensemble is. It can be in one of the following states:

- **NO_ENSEMBLE:** means that the agent is not part of an ensemble;
- **INITIALIZATION:** means that the agent is a part of an ensemble that is in the phase of formation;

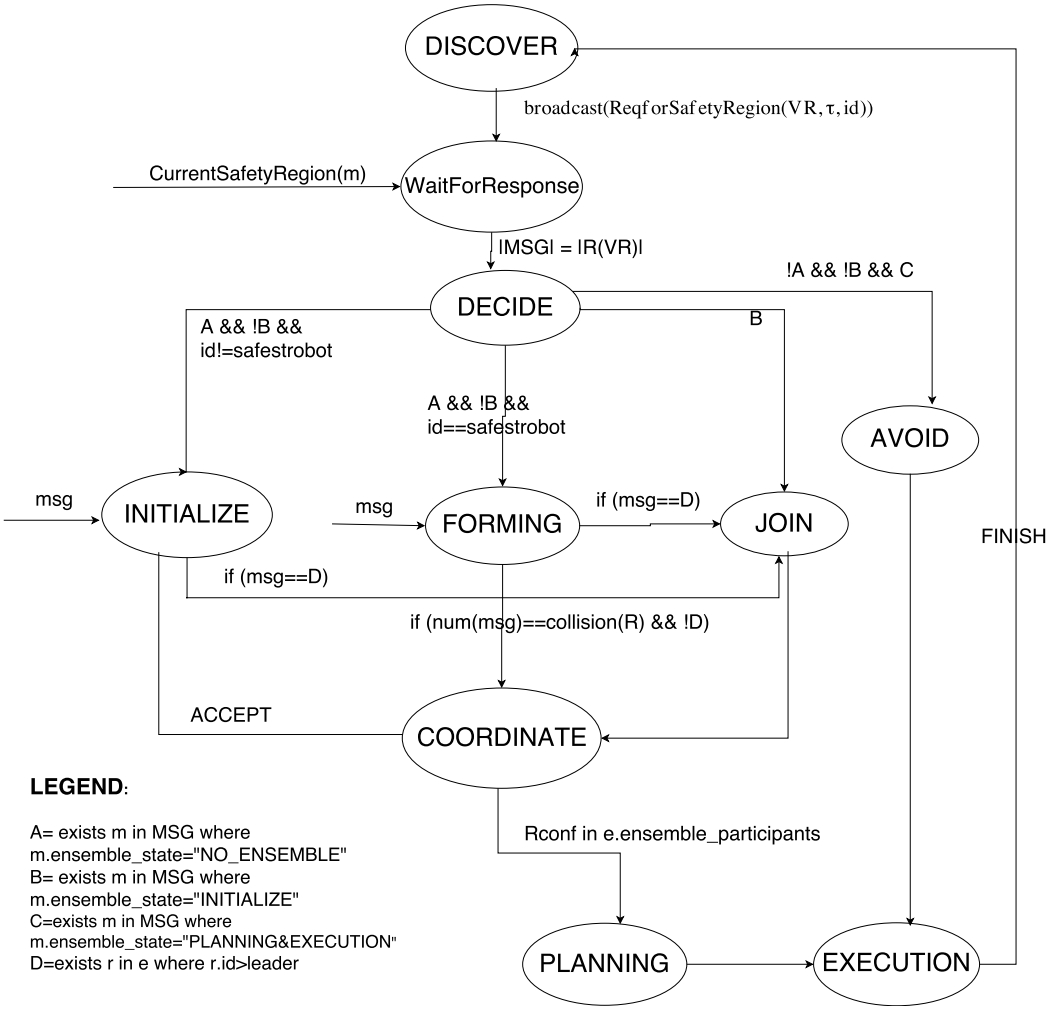


FIGURE 6.11: State machine animating each entity in the safety resolution process

- **PLANNING & EXECUTION:** means that the agent is a part of an ensemble that is established and it executes some safety-related algorithm that can't be interrupted at that time i.e. at the time of execution.

In the *WaitforResponse* state, the agent a_i waits to receive feedback from all the agents that were found in its visible region VR. When it receives message from all the agents in its visible region it goes to a state *Decide*. In the state *Decide*, the agent goes through all messages and for each message it does the following: first, it checks if the agent a_j that sends the message might collide with a_i. If there is a possible collision, it categorizes the message in one of the following categories:

1. **Category 1:** The message is from an agent a_j that is a part of an ensemble in an initialization state (ensemble_state=initialization). In this case, the agent a_i considers the possibility to join to that ensemble.

2. **Category 2:** The message is from an agent a_j that is not part of an ensemble (ensemble_state=no_ensemble). Here, the agent a_i considers the possibility to start with initialization of an ensemble.

3. **Category 3:** The message is from an agent a_j that is a part of an ensemble that is in planning&execution state (ensemble_state=planning&execution). In this case, the agent a_i considers the conflict_region received in the message as a dynamic obstacle and adds it in its collision region $CollisionR$. The ensemble in planning&execution state cannot be interrupted.

After a_i finishes the iteration through all the messages, it does the following decision:

- if there is at least one message from *category 1* (statement B in Figure 6.11 is true), the agent a_i goes to state *JOIN* and initiates the joining process. If there are multiple messages from *category 1*, the agent a_i considers joining to the ensemble that has leader with highest safety index.

- if there are no messages from *category 1* (statement B in Figure 6.11 is false), it checks if there is at least one message from *category 2*, i.e. if statement A in Figure 6.11 is true. If there are multiple messages of that kind, with the function *findsafestrobot* it finds the identifier of the agent with the highest safety index in its visible region. If the agent has the highest safety index in its visible region, it goes into state *FORMING* from where it starts the initialization of the ensemble, otherwise it goes into state *INITIALIZE*.

- This is the case when the agent has received only messages from agents that are parts of ensembles that are in their planning&execution phase (*category 3*). In this case the agent a_i goes into the *AVOID* state where the agent acts as it discovered a dynamic obstacle. In this state the agent should activate some of its solvers for dynamic obstacle avoidance that are able to generate a behavior (solution) for a dynamic obstacle avoidance. Here, the agent performs isolated adaptation and threats this problem on the same level as a dynamic obstacle.

We define conflict set $Rconf$ for an entity e as the set of entities that are in its visible region, can collide with e and are not part of an ensemble that is in planning&execution phase (entities that have send messages from category 2 and category 3). This means that $Rconf$ entities are "open" for performing an appropriate collision avoidance protocol that should avoid the possible collision.

From the *INITIALIZE* state, the entity waits for a message m from the agent that has the highest safety index in its visible region to decide in which state it should go. If it

receives a message that there exists another entity with a higher safety index (statement D is true), it transits to a *JOIN* state from where it starts the procedure for joining an existing ensemble. Otherwise, if it receives an $ACCEPT$ message means that the agent with the highest safety index in its visible region started the process for ensemble creation, and then it goes to a state *Coordinate*. That means that there isn't any other entity with higher safety index in both of their visible regions.
The entity in the *FORMING* state is a possible leader (coordinator) of an ensemble. The entity in this state sends ensemble "proposal" requests to the entities that are in its vision region and have safety regions that overlaps with its safety region at some point of time in the future. The entity e_i stays in the *FORMING* state until one of the following happens:

- it receives a message from an entity in its visible region that "rejects" its ensemble "proposal" meaning that the entity that send the message is aware of another entity that has a higher safety index, so that entity should be the coordinator/leader of the ensemble (statement D is true). In that case, the entity that received the message directly goes into state *JOIN*.
- it receives a message from all entities in $Rconf$ and they all accepted the proposal for ensemble creation. That means statement D is false i.e. there isn't any entity in $Rconf$ that is aware of another entity e_j that is in the *FORMING* state and has a higher safety index.

In the *Coordinate* state, the coordination between the different entities in the ensemble happens. The entity stays in this state, until all entities represented with $Rconf$ are part of the ensemble. When each of $Rconf$ entities join the ensemble, the entity goes into a *Planning* state. The last entity that enters the *Planning* state is the leader. When the leader enters this state, on-the-fly ensemble formation phase finishes and the final ensemble is formed. Here, the leader has information for all ensemble participants and the final conflict region of the ensemble. In this state the leader makes a decision of what is the best solver i.e. what is the best collision avoidance protocol that all ensemble participants should follow. We implemented that by using the function *coordinate*() which is executed only by the ensemble leader (Figure 6.12). Based on the information about the ensemble (conflict regions, participants, etc.), the leader generates a prioritized list of all possible solvers (list of collision avoidance algorithms) embedded in the Knowledge base that could solve the possible collision for this ensemble formation (line 3). Then, the ensemble leader starts from the first algorithm and calls the recursive function $resolve\_safety(A, id, e\_j)$ where A is the algorithm that is chosen for safety resolution, id is the identifier of the entity that executes the algorithm to generate a

solution and e_j is the identifier of the entity that triggered the algorithm in the entity that executes it. Initially, $resolve\_safety(A, id, e\_j)$ is called locally by the ensemble leader so it has the form $resolve\_safety(A, id, id)$ (line 5).

```
1  function coordinate()
2      if (id == ensemble.leader)
3          SolverList = callSolvers(ensemble);
4          foreach A ∈ SolverList:
5              if (resolve_safety(A, id, id))
6                  commit(id)
7                  break
8
9  function resolve_safety(A, id, e_j)
10     S = callSolver(A)
11     Target = derive_coms(ensemble)
12     foreach t ∈ Target:
13         t.solution = rpc (resolve_safety(A, t, id))
14         if (t.solution == false)
15             S = ∅
16             break
17     if (S ≠ ∅)
18         store(S, Target)
19         return true
20     else
21         return false
22
23 function commit(r_id)
24     foreach t ∈ r_id.Target
25         commit(t)
26     update(S)
```

FIGURE 6.12: Code snippets from Planning and Execution state

The function $resolve\_safety$ includes the following. First, the appropriate solver is called to generate a solution S based on the algorithm A (line 10). Then, a corresponding communication is derived using *derive_coms* for each of the reachable entities in the ensemble. In other words, *derive_coms* finds all the children in the problem resolution tree reachable from the corresponding entity in the final ensemble.

For each identified entity, the function $resolve\_safety$ is called with the algorithm A (line 13) and a solution S is calculated.

The function $resolve\_safety$ returns a boolean:

- it returns true when the entity and all its children in the problem resolution tree contain the algorithm A in its knowledge base. If that is the case, the generated solution S is stored locally (function store (line 18));
- it returns false when the entity itself or one of its children (targets) doesn't contain the algorithm A in its knowledge base.

In the end the $resolve\_safety$ for the ensemble leader (line 5) returns true if all the entities in the ensemble are able and agreed to perform some algorithm A. If that is not

the case, the ensemble leader goes through the other algorithms in the *SolverList* and repeats the process until finds a suitable algorithm A for which all ensemble participants are able to perform it. If it finds one, it calls locally the function *commit* and breaks the loop. Execution of the commit function is when the entity enters the *Execution* state in Figure 6.11.

The *commit* function lines 23-26. is recursive, it goes through each entity in the ensemble and enacts a distributed commit of the best solution. It updates the solution S in each entity by updating its behaviour tree with the function update(S) (line 26). When the entity finishes with the *commit* function in the *Execution* state, it goes to *Discover*.

6.6.2 Correctness and completeness

In order to prove correctness of the safety resolution approach we need to prove the correctness of the safety resolution algorithm. In order to prove the correctness of the safety resolution resolution algorithm we need to analyze: (i) the protocol for on-the-fly ensemble formation and (ii) the recursive function for selecting and committing a solution. We define correctness as follows.

Theorem 6.14 (Correctness)**.** *If an entity e triggers a safety problem P_i, then the Safety Problem Resolution Algorithm computes a solution S_i where the safety invariants in Definition 6.13 are always satisfied.*

Proof. Our assumption is that each of the entities in the safety collective adaptation process contains at least one safety solver (collision avoidance protocol) that is compatible to the solvers of the other entities in the ensemble. This means that we assume that when an ensemble is formed, each of the entities participating in the ensemble contains an appropriate solver that can generate a solution that will satisfy the safety invariant defined in Definition 6.13. Therefore, to guarantee correctness of the safety resolution process we just need to prove correctness of the protocol for on-the-fly ensemble formation.

In order to prove the correctness of the protocol for on-the-fly ensemble formation, we verify (using model-checking) the following properties about the coordination protocol: (1) at each point of time there won't be any ensembles that are in *planning&execution* phase and that have overlapping $conflict\_region\ CR$; (2) the highest safety index computed by the safety resolution algorithm is consistent across all entities in an ensemble that is in *planning&execution* phase. We formally specify them as follows. We denote with T the total mission execution time and with E^{τ}_{pe} the set of all ensembles in the system that are in *planning&execution* phase at time τ. Hence, we define (1) as:

$\forall \tau \in T \&\& \forall es_i, es_j \in E^{\tau}_{pe}; es_i.CR \cap es_j.CR = \emptyset$ and we define (2) for an $es \in E^{\tau}_{pe}$ as: $\forall e_i, e_j \in es; e_i.s\_id = e_j.s\_ = id$.

However, the safety resolution algorithm by itself is not complete due to the following reason: each entity in a stable ensemble might not have a solver compatible to the solver proposed by the leader. However, for completeness of the approach, as we mentioned earlier, we assume that each entity in the system has at least one safety solver that is compatible to the solvers of the rest of the system. We consider this assumption reasonable because we consider safety independently from the mission and in this way, we can reason about safety before the start of the mission. That is how we guarantee safety during the whole mission execution.

□

6.7 Related work

In this section we review recent work on run-time modelling for mobile multi-robot systems. We investigate what kind of run-time modeling approaches are proposed for the different multi-robot systems and in which type of environments they are applicable. Furthermore, we focus at identifying and classifying approaches that address safety. Our aim is to understand how safety is managed and what are the constraints and limitations of existing methodologies addressing safety. Additionally, we consider if the approach supports adaptation on the system and which adaptation decisions could be made at run-time.

In Table 6.2, we note some of these run-time modeling approaches and categorize them according to the following criteria:

1. System & environment characteristics;
2. Safety management;
3. Adaptation mechanisms.

For each criteria we identify several parameters. For each of them, we categorize the approaches as follows.

System & environment characteristics

- **Agent type**: whether the considered agents in the system are homogeneous or heterogeneous.

Work	System & environment characteristics						Safety Management		Adaptation mechanisms		
	A. Type	Openness	Ens. type	Mgmt.	Hierarchy	ENV	Coop. Mechanisms	Concerns sep.	Adapt	Type	Human
[107]	Heterogenous	N/A	dynamic	N/A	N/A	N/A	cooperative	NO	YES	BOTH	NO
[108]	Heterogenous	YES	N/A	N/A	NO	dynamic	local	NO	YES	isolated	NO
[109]	Heterogenous	NO	N/A	N/A	N/A	dynamic	cooperative	NO	NO	N/A	N/A
[110]	Heterogenous	NO	dynamic	distributed	NO	static	cooperative	NO	YES	isolated	NO
[111]	Homogeneous	NO	N/A	N/A	NO	static	local	NO	YES	isolated	NO
[112]	Homogeneous	NO	N/A	N/A	NO	static	cooperative	NO	NO	N/A	NO
[113]	Heterogenous	NO	N/A	N/A	NO	dynamic	centralized	NO	YES	N/A	N/A
[114]	Homogeneous	NO	dynamic	centralized	NO	N/A	centralized	NO	NO	N/A	N/A
[115]	Heterogeneous	YES	dynamic	distributed	YES	dynamic	N/A	N/A	YES	BOTH	NO

TABLE 6.2: Literature Comparison

- **Openness**: whether the approach supports systems that can accept external agents at run-time (e.g., new robots entering the mission).
- **Ensemble type**: whether the proposed approach supports formation of an ensemble structure (group of agents) that can change at run-time (dynamic) or not (static).
- **Ensemble management**: whether an ensemble is managed in a centralised or in a distributed way.
- **Hierarchy**: whether the approach provides a mechanism for hierarchical structure of ensembles (e.g. an ensemble of ensembles).
- **Environment** (dynamic vs. static): whether the approach supports modeling of systems that operate in environment that can change at run-time (e.g., moving obstacles, some other elements outside of the system can change their status).

Safety Management

- **Cooperation mechanisms**: whether the approach allows safety mechanisms that involve cooperation between different agents rather than centralized management of safety entity or local management on single robots, without any cooperation.

  It is *LOCAL* if safety mechanisms are conceived to work on single robots, without any cooperation, *CENTRALIZED* if the knowledge of the overall system is maintained by a centralized entity, or *COOPERATIVE* if there are mechanisms to share knowledge between different robots that take part in the mission.
- **Separation of concerns**: whether the approach keeps the management of safety-specific issues (e.g. safety rules) separated from the management of mission-specific issues.

Adaptation decisions

- **Adaptability**: whether the approach supports MMRSs that can adapt.

- **Type**: whether the approach supports a collective adaptation where a collection of autonomous agents collaborate together to satisfy a particular goal or isolated adaptation where one agent adapts independently from the rest of the system.
- **Human Controllability**: whether the approach enables an operator (human) to be involved in the adaptation process.

As shown in Table 6.2, most of the approaches are unable to deal with open systems (only 2/9 approaches are able to deal with open systems). By open systems, we mean systems that can accept external entities at run-time (e.g., new robots or new human actors). This implies that most of the approaches that have been proposed do not consider that the system evolves in terms of addition or removal of robots and/or other types of agents, including humans. This is indeed an interesting research direction since systems of the near future will be necessarily characterized by openness, and it is often impossible to assess at design time the exact boundaries and topology of the system.

A peculiar system characteristic is the capability of managing teams consisting of robots of different types (e.g., robots for grabbing objects, for video streaming, sensing and discovering relevant information). According to Table 6.2 most of the analyzed systems have the capability of managing heterogeneous robots, which is the direction in which we are going with our approach.

In order to manage different unpredictable situations of missions and considering situations where there is only partial communication between different agents, it is preferable that the MMR system is capable of grouping and regrouping agents in ensembles at run-time in a decentralized fashion. Most of the approaches propose solutions where they assume that all robots in the system will be able to communicate to each other. Only 2/9 approaches discuss about the possible benefits of distributed ensemble formation. The concept of ensemble enables single agents to take part in a group where they will follow certain rules and in return the ensemble offers certain advantages with respect to a preservation of a particular system quality. In this context, we don't need to consider the whole system to analyze if a particular system quality is satisfied, we only consider part of it.

Furthermore, as shown in Table 6.2, in all approaches the management of safety-specific issues (e.g., safety rules) is not kept separated from the functional management of the robots (e.g., the mission). Keeping a separation of concerns means for instance that the approach prescribes a special layer for managing safety, which is totally separated from the rest of the system.

Regarding the cooperation mechanisms, most of the approaches adopt local safety mechanisms, i.e. safety mechanisms that are conceived to work on single robots, without any

cooperation. Centralized safety management mechanism means that there exist an entity managing the safety aspect of the overall system. As can be seen in Table 6.2 only 2 approach have a centralized safety management mechanism. Instead, 4 approaches rely on co-operative safety mechanisms, meaning that safety mechanisms involve a cooperation between different robots.

Regarding the adaptation mechanisms, most of the approaches allow the system to adapt at run-time, meaning that the system is able to adapt (e.g., behaviour adaptation, trajectory recalculation, goal renegotiation, etc.) in order to find a solution depending on some change in the context. Adaptability might be considered in conjunction with context awareness since awareness of the context is a required capability in order to support adaptability. Human involvement in the adaptation process brings some degree of control-ability that can help in defining regulations and rules about the responsibilities of the operators in operational scenarios and make adaptation more practical and safer. None of the approaches in Table 6.2 includes the human as a factor in the adaptation process. Regarding the collectiveness of the adaptation process, 3 of the approaches consider isolated adaptation where the agent adapts its behaviour independently from the rest of the system, while only 2 approaches consider the two types of adaptation: (i) on a collective level where multiple agents must adapt altogether and transactionally and (ii) isolated adaptation where one agent adapts its behaviour independently from the rest of the system.

Classifying our framework in this classification schema is as follows: It supports modeling of *open systems* which consist of *heterogeneous agents* that are *context-aware* about their operational context. The agents can be grouped in *dynamic ensembles*, which are groups of agents that are managed in a *distributed way* among the ensemble participants. Furthermore, the framework allows for a *hierarchical structure of ensembles* for satisfying a particular goal as we showed in [4]. Regarding the environment, the framework has mechanisms for modeling MMRSs that operate in *dynamic environments* and have some degree of *unpredictability* (ex. birds flying, animals walking, etc).

Most of the frameworks we have analyzed do not consider safety aspects separate from the functional behaviour of the robots. In our framework we make a *clear separation of concerns between safety and mission* aspects. We consider this as extremely important for managing complex missions and it is one of the fundamental parts on which we base our framework. This way an operator modeling its mission can focus on the mission specification, while a safety engineer can focus on the safety-specific mechanisms, thus making safety-specific mechanisms reusable across missions, projects, and organizations.

Another really important feature of our framework is allowing *cooperation mechanisms* between different agents when safety-related issues are triggered. This means that safety

is not managed in a centralized way (there isn't an entity that manages the whole aspect of safety, but it is managed on a level of ensemble in a distributed way where each agent should perform an appropriate behavior as part of the ensemble).

Moreover, our modeling framework contains structures that enable system designers to design systems which are *adaptable* during mission execution. It allows part of the adaptation decisions to be done at *run-time*. That being said, we make clear distinction about which decisions should be made at design-time versus decisions at run-time. Regarding the type, we allow agents to adapt on a collective (ensemble) level, which means that a collection of autonomous agents collaborate to perform adaptation in order to satisfy a particular goal or solve a particular problem. In the end, we allowed the operator (human) be able to have control in the adaptation process. Some adaptation decisions can't be done by the system at run-time, however with our approach as we showed in [4] we allow the operator to take over and participate as agent in the adaptation process.

6.8 Conclusions

In this chapter, we presented two collective adaptation approaches: one for (potentially partial) resolution of mission problems and one for full safety resolution i.e. an approach that ensures a full satisfaction of safety invariants. While most of the proposed solutions for collective adaptation work under the assumption that all the knowledge used to adapt a system is fully specified at design time (i.e., a predefined set of issues) and is centrally controlled by a specific component (i.e., a set of predefined solvers), our approaches, as depicted previously, address collective adaptation problems in a decentralized fashion, at run-time, with new solvers that can be introduced at any time. At the same time, in highly dynamic and distributed environments, our approaches provide a way to dynamically understand which parts of the system should be selected to help solve an adaptation issue making a clear distinction when resolving mission vs. safety-related issues. That way, we can ensure full satisfaction of safety, while guaranteeing (potentially partial) mission completion.

Chapter 7

Conclusion and Future Work

7.1 Conclusions

MMRSs are facing a number challenges preventing them from being used in our everyday tasks: (i) robots need to be able to operate in unknown environments, (ii) be required to collaborate with each other and even with humans to accomplish complex missions, and (iii) despite failures or malfunctions, robots should never injure people or create loss or severe damage to equipment/property. Those challenges are getting targeted from various research perspectives, producing a fragmented research landscape.

From the aformentioned challenges the following research question emerged: ***"How to preserve safety while guaranteeing (potentially partial) mission completion for MMRSs?"***

In this thesis we approches this research question by decomposing it on two sub-research questions:

R1. How to create a clear separation of concerns in modeling safety and mission aspects for MMRSs?

To address this question we designed a modeling framework that supports specification and execution of missions in MMRSs distinguishing between mission-related and safety-related behaviour mechanisms.

R.2.How to guarantee safety and (potentially partial) mission completion for MMRSs in relation to operational context changes at run-time?

To address this question we designed a formal model that specifies the collective adaptation behavior of entities in MMRSs and provided two approaches: a mission problem

resolution approach for (potentially partial) resolution of mission problems and a safety resolution approach that ensures a full satisfaction of safety invariants.

Next, we will list all the contributions in this thesis and provide research directions for future work.

7.2 Contributions

The main contributions of this thesis are:

- providing a complete, comprehensive and replicable picture of the state of the art on safety for mobile robotic systems (MSRs) using systematic mapping methodology (Chapter 3);
- designing a modeling framework that supports specification and execution of missions in MMRSs and ensures safety and guarantees (potentially partial) mission completion (Chapter 4, Chapter 5);
- a formal model that specify the collective adaptation behavior of entities in MMRSs in an operational context considering safety as first class element and humans as actors in the adaptation process (Chapter 5);
- a generic approach for managing collective adaptation of MMRSs in a decentralized fashion at run-time (Chapter 5);
- a specific mission problem resolution approach for managing collective decentralized run-time adaptation of MMRSs guaranteeing (potentially partial) mission completion (Chapter 6);
- a specific safety problem resolution approach for managing collective decentralized run-time adaptation of MMRSs ensuring satisfaction of safety invariants (Chapter 6)

7.3 Future Work

In this section we highlight some research directions for future work.

As future work we are planning to integrate our framework with a suitable extension of the FLYAQ platform [3, 61]. This platform permits to graphically define civilian missions for a team of autonomous multicopters via a domain specific language to make the specification of missions accessible to people with no expertise in IT and robotics.

Moreover, we plan to further experiment the approach in different application domains and in real environments. We will carry on a experimental campaign to evaluate our framework both by simulation and with real deployments. Simulation will be performed by using a Software-In-The- Loop (SITL) platform.

We plan to perform an experiment to analyze the collective adaptation process (CAP). We plan to measure mission satisfiability that gives information on how much percentage of the mission is performed taking in consideration various application domains. We will be using heterogeneous robots operating under various circumstances (e.g. different number of tasks to be performed, different size of MMRSs, different number of problems triggered during mission execution, different ratio between safety and mission problems triggered during mission execution etc.). The final goal is to find out if our CAP is scalable for managing real-sized missions.

Some adaptation decisions can't be done by the system at run-time, so we allowed the operator (human) to be able to have control in the adaptation process. As we showed in Chapter 5 we allow the operator to take over and participate as agent in the adaptation process. As future work, we want to better understand the impact of human involvement in the adaptation process. We want to explore the possibilities where the human is first-class actor in the system (self-controlled adaptation) and how the adaptation process would look like if we model the human behavior separated from the other entities of the system.

Bibliography

[1] Darko Bozhinoski. Managing safety and adaptability in mobile multi-robot systems. In *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, pages 135–140. ACM, 2015.

[2] Darko Bozhinoski, Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, and Ivica Crnkovic. Safety for mobile robotic systems: a systematic mapping study. under submission to Journal of Systems and Software (JSS), 2017.

[3] Darko Bozhinoski, Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, and Massimo Tivoli. Flyaq: Enabling non-expert users to specify and generate missions of autonomous multicopters. In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 801–806. IEEE, 2015.

[4] Darko Bozhinoski, Antonio Bucchiarone, Ivano Malavolta, Annapaola Marconi, and Patrizio Pelliccione. Leveraging collective run-time adaptation for uav-based systems. In *Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on*, pages 214–221. IEEE, 2016.

[5] Darko Bozhinoski, David Garlan, Ivano Malavolta, and Pelliccione Patrizio. Managing safety and mission completion viacollective run-time adaptation, 2017. under submission to Journal of Systems Architecture: Embedded Software Design (JSA).

[6] Introduction to robots. http://www.galileo.org/robotics/intro.html, October 2014.

[7] IEEE Society of Robotics and Automations Technical Committee on Networked Robots. URL http://www-users.cs.umn.edu/~isler/tc/.

[8] Martin Ford. Rise of the robots: Technology and the threat of a jobless future. *Organization Management Journal*, 13(2):115–117, 2016. doi: 10.1080/15416518.2016.1180076.

[9] Henning Kagermann, Wolfgang Wahlster, and Johannes Helbig. Recommendations for implementing the strategic initiative industrie 4.0 – securing the future of german manufacturing industry. Final report of the industrie 4.0 working group,

München, 2013. URL http://forschungsunion.de/pdf/industrie_4_0_final_report.pdf.

[10] Elena Garcia, María A Jimenez, Pablo Gonzalez De Santos, and Manuel Armada. The evolution of robotics research. *Robotics & Automation Magazine, IEEE*, 14 (1):90–103, 2007.

[11] T. Skrzypietz. *Unmanned Aircraft Systems for Civilian Missions*. BIGS policy paper: Brandenburgisches Institut für Gesellschaft und Sicherheit. BIGS, 2012. URL http://books.google.se/books?id=jQVPmwEACAAJ.

[12] Darko Bozhinoski, Federico Ciccozzi, Ivica Crnkovic, Davide Di Ruscio, Ivano Malavolta, and Patrizio Pelliccione. Mobile multi-robot systems in everyday life: Technological challenges and opportunities. under submission to IEEE Software, 2014.

[13] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[14] Patricia Cronin, Frances Ryan, and Michael Coughlan. Undertaking a literature review: a step-by-step approach. *British Journal of Nursing*, 17(1):38, 2008.

[15] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33:2004, 2004.

[16] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering*. Computer Science. Springer, 2012.

[17] Darko Bozhinoski, Ivano Malavolta, Antonio Bucchiarone, and Annapaola Marconi. Sustainable safety in mobile multi-robot systems via collective adaptation. In *Self-Adaptive and Self-Organizing Systems (SASO), 2015 IEEE 9th International Conference on*, pages 172–173. IEEE, 2015.

[18] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7 (1):1–41, 2013. ISSN 1935-3812.

[19] Safety standards: International organization for standardization (iso) for robots and robot systems integration. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_tc_browse.htm?commid=54138, October 2014.

[20] Henrik I Christensen, Antonio Bicchi, and Domenico Prattichizzo. *Control Problems in Robotics*. Springer, 2003.

[21] Shengyong Chen, Youfu Li, and Ngai Ming Kwok. Active vision in robotic systems: A survey of recent developments. *The International Journal of Robotics Research*, 30(11):1343–1377, 2011.

[22] Ayssam Elkady and Tarek Sobh. Robotics middleware: A comprehensive literature survey and attribute-based bibliography. *Journal of Robotics*, 2012, 2012.

[23] Grzegorz Chmaj and Henry Selvaraj. Distributed processing applications for uav/-drones: a survey. In *Progress in Systems Engineering*, pages 449–454. Springer, 2015.

[24] Howie Choset. Coverage for robotics–a survey of recent results. *Annals of mathematics and artificial intelligence*, 31(1-4):113–126, 2001.

[25] Michael Thielscher. *Reasoning robots: the art and science of programming robotic agents*, volume 33. Springer, 2006.

[26] Davide Brugali and Erwin Prassler. Software engineering for robotics. *Robotics & Automation Magazine, IEEE*, 16(1):9–15, 2009.

[27] Christopher W. Johnson. What are emergent properties and how do they affect the engineering of complex systems? *Reliability Engineering & System Safety*, 91(12): 1475 – 1481, 2006. ISSN 0951-8320. doi: http://dx.doi.org/10.1016/j.ress.2006.01.008. URL http://www.sciencedirect.com/science/article/pii/S0951832006000330. Complexity in Design and Engineering.

[28] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161–180, 2010.

[29] Michael A. Goodrich and Alan C. Schultz. Human-robot interaction: A survey. *Found. Trends Hum.-Comput. Interact.*, 1(3):203–275, January 2007. ISSN 1551-3955. doi: 10.1561/1100000005. URL http://dx.doi.org/10.1561/1100000005.

[30] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2008.

[31] Algirdas Avižienis, Jean-Claude Laprie, and Brian Randell. Dependability and its threats: a taxonomy. In *Building the Information Society*, pages 91–120. Springer, 2004.

[32] Nary Subramanian and Lawrence Chung. Metrics for software adaptability. *Proc. Software Quality Management (SQM 2001), April*, 2001.

[33] B. Hailpern and P. Tarr. Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal*, 45(3):451–461, 2006. ISSN 0018-8670. doi: 10.1147/sj.453.0451.

[34] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[35] Barbara A Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE-2007-01, Keele University and University of Durham, 2007.

[36] Oxford dictionary: Origin of the term robot. https://www.oxforddictionaries.com/definition/english/robot, October 2014.

[37] Tom Harris. How robots work - howstuffworks, 2014. URL http://science.howstuffworks.com/robot.htm.

[38] Olesya Ogorodnikova. *Human Robot Interaction: The Safety Challenge*. PhD thesis, Budapest University of Technology and Economics, 2010.

[39] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley, 1994.

[40] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571–583, 2007.

[41] Tore Dyba, T. Dingsoyr, and G.K. Hanssen. Applying systematic reviews to diverse study types: An experience report. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 225–234, Sept 2007. doi: 10.1109/ESEM.2007.59.

[42] Staffs Keele. Guidelines for performing systematic literature reviews in software engineering. Technical report, Technical report, EBSE Technical Report EBSE-2007-01, 2007.

[43] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 2002. ISBN 0133056996.

[44] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International*

Conference on Evaluation and Assessment in Software Engineering, EASE '14, pages 38:1–38:10, New York, NY, USA, 2014. ACM.

[45] Samireh Jalali and Claes Wohlin. Systematic literature studies: database searches vs. backward snowballing. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 29–38. ACM, 2012.

[46] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77, Swinton, UK, UK, 2008. British Computer Society.

[47] Martin Ivarsson and Tony Gorschek. A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering*, 16(3):365–395, 2011.

[48] *ISO/IEC/IEEE 42010, Systems and software engineering — Architecture description.* ISO, December 2011.

[49] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25, 2006.

[50] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering*, 11(1):102–107, 2006. ISSN 0947-3602. doi: 10.1007/s00766-005-0021-6.

[51] John C Mankins. Technology readiness levels. *White Paper, April*, 6, 1995.

[52] Thomas D Cook, Donald Thomas Campbell, and Arles Day. *Quasi-experimentation: Design & analysis issues for field settings*, volume 351. Houghton Mifflin Boston, 1979.

[53] Paolo Di Francesco, Ivano Malavolta, and Patricia Lago. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *Software Architecture (ICSA), 2017 IEEE International Conference on*, pages 21–30. IEEE, 2017.

[54] Smart robots market - analysis & forecast to 2020, report of april 2015. Technical report, 2015. URL http://www.marketsandmarkets.com/Market-Reports/smart-robots-market-48470534.html.

[55] EU H2020. Robotics 2020 Multi-Annual Roadmap For Robotics in Europe - http://sparc-robotics.eu/wp-content/uploads/2014/05/H2020-Robotics-Multi-Annual-Roadmap-ICT-2016.pdf. 2016.

[56] F. Ciccozzi, D. Di Ruscio, I. Malavolta, P. Pelliccione, and J. Tumova. Engineering the software of robotic systems. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 507–508, May 2017. doi: 10.1109/ICSE-C.2017.167.

[57] Tadele Shiferaw Tadele, Theo JA Vries, and Stefano Stramigioli. The safety of domestic robotics: A survey of various ssafety-related publications. *IEEE Robotics & Automation Magazine*, 21(3):134–142, 2014.

[58] Michael A Goodrich and Alan C Schultz. Human-robot interaction: a survey. *Foundations and trends in human-computer interaction*, 1(3):203–275, 2007.

[59] Milos Vasic and Aude Billard. Safety issues in human-robot interactions. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 197–204. IEEE, 2013.

[60] R Alami, A Albu-Schaeffer, A Bicchi, R Bischoff, R Chatila, A De Luca, A De Santis, G Giralt, J Guiochet, G Hirzinger, et al. Safe and dependable physical human-robot interaction in anthropic domains: State of the art and challenges. In *Proc. IROS*, volume 6. Citeseer, 2006.

[61] Davide Di Ruscio, Ivano Malavolta, and Patrizio Pelliccione. Engineering a platform for mission planning of autonomous and resilient quadrotors. In *International Workshop on Software Engineering for Resilient Systems*, pages 33–47. Springer, 2013.

[62] Steven S Skiena. The algorithm design manual, 1997. *Stony Brook, NY: Telos Pr*, 504.

[63] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. In *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009*, pages 65–100. Springer, 2009.

[64] Mojtaba Nouri Bygi and Mohammad Ghodsi. 3d visibility graph. *Computational Science and its Applications, Kuala Lampur*, 2007.

[65] Seuk-Woo Ryu, Young-ho Lee, Tae-Yong Kuc, Sang-Hun Ji, and Yong-Seon Moon. A search and coverage algorithm for mobile robot. In *Ubiquitous Robots and*

Ambient Intelligence (URAI), 2011 8th International Conference on, pages 815–821. IEEE, 2011.

[66] Raphael Mannadiar and Ioannis Rekleitis. Optimal coverage of a known arbitrary environment. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 5525–5530. IEEE, 2010.

[67] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013.

[68] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Science of computer programming*, 72(1):31–39, 2008.

[69] Davide Di Ruscio, Ivano Malavolta, and Patrizio Pelliccione. The role of parts in the system behaviour. In *International Workshop on Software Engineering for Resilient Systems*, pages 24–39. Springer, 2014.

[70] Christopher W Johnson. What are emergent properties and how do they affect the engineering of complex systems?, 2006.

[71] Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, and Massimo Tivoli. Automatic generation of detailed flight plans from high-level mission descriptions. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 45–55. ACM, 2016.

[72] A. Bucchiarone, C. Mezzina, M. Pistore, H. Raik, and G. Valetto. Collective adaptation in process-based systems. In *SASO 2014*, pages 151–156, 2014.

[73] Thomas L Saaty. *What is the analytic hierarchy process?* Springer, 1988.

[74] Piergiorgio Bertoli, Raman Kazhamiakin, Massimo Paolucci, Marco Pistore, Heorhi Raik, and Matthias Wagner. Control flow requirements for automated service composition. In *ICWS 2009*, pages 17–24, 2009.

[75] Dayong Ye, Minjie Zhang, and Danny Sutanto. Self-adaptation-based dynamic coalition formation in a distributed agent network: A mechanism and a brief survey. *IEEE Trans. Parallel Distrib. Syst.*, 24(5):1042–1051, 2013.

[76] Radu-Casian Mihailescu, Matteo Vasirani, and Sascha Ossowski. Dynamic coalition adaptation for efficient agent-based virtual power plants. In *Multiagent System Technologies - 9th German Conference, MATES 2011, Berlin, Germany, October 6-7, 2011. Proceedings*, volume 6973 of *Lecture Notes in Computer Science*, pages 101–112. Springer, 2011.

[77] T. Pinto, H. Morais, P. Oliveira, Z. Vale, . Praa, and C. Ramos. A new approach for multi-agent coalition formation and management in the scope of electricity markets. *Energy*, 36(8):5004–5015, 2011.

[78] Mark Sims, Claudia V. Goldman, and Victor R. Lesser. Self-organization through bottom-up coalition formation. In *AAMAS*, pages 867–874, 2003.

[79] Nima Roohi and Gwen Salaün. Realizability and dynamic reconfiguration of chor specifications. *Informatica (Slovenia)*, 35(1):39–49, 2011.

[80] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Self-adaptive monitors for multiparty sessions. In *22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2014, Torino, Italy, February 12-14, 2014*, pages 688–696. IEEE, 2014.

[81] Mila Dalla Preda, Maurizio Gabbrielli, Saverio Giallorenzo, Ivan Lanese, and Jacopo Mauro. Developing correct, distributed, adaptive software. *Sci. Comput. Program.*, 97:41–46, 2015.

[82] Rocco De Nicola, Michele Loreti, Rosario Pugliese, and Francesco Tiezzi. A formal approach to autonomic systems programming: The SCEL language. *TAAS*, 9(2): 7, 2014.

[83] Rolf Hennicker and Annabelle Klarl. Foundations for ensemble modeling - the helena approach - handling massively distributed systems with elaborate ensemble architectures. In *Specification, Algebra, and Software*, pages 359–381, 2014.

[84] Stefan Niemczyk and Kurt Geihs. Adaptive run-time models for groups of autonomous robots. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2015 IEEE/ACM 10th International Symposium on*, pages 127–133. IEEE, 2015.

[85] Christopher Zhong and Scott A DeLoach. Runtime models for automatic reorganization of multi-robot systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 20–29. ACM, 2011.

[86] Chih-Han Yu, Justin Werfel, and Radhika Nagpal. Collective Decision-making in Multi-agent Systems by Implicit Leadership. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 3*, AAMAS '10, pages 1189–1196. International Foundation for Autonomous Agents and Multiagent Systems, 2010. ISBN 0-98265-713-7.

[87] Chih-Han Yu and R. Nagpal. Self-adapting modular robotics: A generalized distributed consensus framework. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 1881–1888, 2009.

[88] Danny Weyns, Robrecht Haesevoets, and Alexander Helleboogh. The MACODO organization model for context-driven dynamic agent organizations. *TAAS*, 5(4): 16, 2010.

[89] F.A. de Oliveira, T. Ledoux, and R. Sharrock. A Framework for the Coordination of Multiple Autonomic Managers in Cloud Environments. In *Self-Adaptive and Self-Organizing Systems (SASO), 2013 IEEE 7th International Conference on*, pages 179–188, 2013.

[90] Vincenzo Grassi, Moreno Marzolla, and Raffaela Mirandola. Qos-aware fully decentralized service assembly. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 53–62. IEEE Press, 2013.

[91] George Edwards, Joshua Garcia, Hossein Tajalli, Daniel Popescu, Nenad Medvidovic, Gaurav Sukhatme, and Brad Petrus. Architecture-driven Self-adaptation and Self-management in Robotics Systems. In *Proceedings of SEAMS '09*, pages 142–151, Washington, DC, USA, 2009. IEEE Computer Society.

[92] Daniel Sykes, Jeff Magee, and Jeff Kramer. FlashMob: Distributed Adaptive Self-assembly. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, pages 100–109. ACM, 2011.

[93] Linas Laibinis, Inna Pereverzeva, and Elena Troubitsyna. Formal reasoning about resilient goal-oriented multi-agent systems. *Science of Computer Programming*, 2017.

[94] Tomasz P. Michalak, Jacek Sroka, Talal Rahwan, Michael Wooldridge, Peter McBurney, and Nicholas R. Jennings. A distributed algorithm for anytime coalition structure generation. In Wiebe van der Hoek, Gal A. Kaminka, Yves Lesprance, Michael Luck, and Sandip Sen, editors, *AAMAS*, pages 1007–1014. IFAAMAS, 2010. ISBN 978-0-9826571-1-9.

[95] Sarvapali D. Ramchurn, Maria Polukarov, Alessandro Farinelli, Cuong Truong, and Nicholas R. Jennings. Coalition formation with spatial and temporal constraints. In Wiebe van der Hoek, Gal A. Kaminka, Yves Lesprance, Michael Luck, and Sandip Sen, editors, *AAMAS*, pages 1181–1188. IFAAMAS, 2010.

[96] Hyung Jun Ahn, Habin Lee, and Sung Joo Park. A flexible agent system for change adaptation in supply chains. *Expert Syst. Appl.*, 25(4):603–618, 2003.

[97] Nicholas R. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001.

[98] Michele Colledanchise and Petter Ögren. How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *IEEE Transactions on Robotics*, 33(2):372–389, 2017.

[99] R Geoff Dromey. From requirements to design: Formalizing the key steps. In *Software Engineering and Formal Methods, 2003. Proceedings. First International Conference on*, pages 2–11. IEEE, 2003.

[100] Alejandro Marzinotto, Michele Colledanchise, Christian Smith, and Petter Ögren. Towards a unified behavior trees framework for robot control. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 5420–5427. IEEE, 2014.

[101] Morteza Lahijanian, Matthew R Maly, Dror Fried, Lydia E Kavraki, Hadas Kress-Gazit, and Moshe Y Vardi. Iterative temporal planning in uncertain environments with partial satisfaction guarantees. *IEEE Transactions on Robotics*, 32(3):583–599, 2016.

[102] Antonio Bucchiarone, Claudio Antares Mezzina, Marco Pistore, Heorhi Raik, and Giuseppe Valetto. Collective adaptation in process-based systems. In *Self-Adaptive and Self-Organizing Systems (SASO), 2014 IEEE Eighth International Conference on*, pages 151–156. IEEE, 2014.

[103] Tomas Bures, Filip Krijt, Frantisek Plasil, Petr Hnetynka, and Zbynek Jiracek. Towards intelligent ensembles. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, page 17. ACM, 2015.

[104] P language Manual. URL https://github.com/p-org/P/blob/master/Doc/Manual/pmanual.pdf.

[105] Tony Andrews, Shaz Qadeer, Sriram Rajamani, Jakob Rehof, and Yichen Xie. Zing: A model checker for concurrent software. In *Computer Aided Verification*, pages 28–32. Springer, 2004.

[106] Martin Wirsing, Matthias Hölzl, Nora Koch, and Philip Mayer. *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*, volume 8998. Springer, 2015.

[107] Yanzhe Cui, Richard M Voyles, Joshua T Lane, and Mohammad H Mahoor. Refresh: A self-adaptation framework to support fault tolerance in field mobile robots. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1576–1582. IEEE, 2014.

[108] Lynne E Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE transactions on robotics and automation*, 14(2):220–240, 1998.

[109] Márcio G Morais, Felipe R Meneguzzi, Rafael H Bordini, and Alexandre M Amory. Distributed fault diagnosis for multiple mobile robots using an agent programming language. In *Advanced Robotics (ICAR), 2015 International Conference on*, pages 395–400. IEEE, 2015.

[110] M Tahir Khan, MU Qadir, F Nasir, and CW de Silva. A framework for a fault tolerant multi-robot system. In *Computer Science & Education (ICCSE), 2015 10th International Conference on*, pages 197–201. IEEE, 2015.

[111] Eduardo Castello, Tomoyuki Yamamoto, Fabio Dalla Libera, Wenguo Liu, Alan FT Winfield, Yutaka Nakamura, and Hiroshi Ishiguro. Adaptive foraging for simulated and real robotic swarms: the dynamical response threshold approach. *Swarm Intelligence*, 10(1):1–31, 2016.

[112] Ankush Desai, Indranil Saha, Jianqiao Yang, Shaz Qadeer, and Sanjit A Seshia. Drona: a framework for safe distributed mobile robotics. In *Proceedings of the 8th International Conference on Cyber-Physical Systems*, pages 239–248. ACM, 2017.

[113] Jonathan A DeCastro, Javier Alonso-Mora, Vasumathi Raman, Daniela Rus, and Hadas Kress-Gazit. Collision-free reactive mission and motion planning for multi-robot systems. In *Robotics Research*, pages 459–476. Springer, 2018.

[114] Arjav Desai, Ellen A Cappo, and Nathan Michael. Dynamically feasible and safe shape transitions for teams of aerial robots. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 5489–5494. IEEE, 2016.

[115] Yehia Abd Alrahman, Rocco De Nicola, and Michele Loreti. On the power of attribute-based communication. In *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, pages 1–18. Springer, 2016.