Decentralised Control for Distributed Self-adaptive Systems with Strict Quality-of-Service Requirements

Saud Yonbawi

PhD

University of York
Computer Science

March 2021

Abstract

Many systems from domains such as the cyber-physical systems (CPS) and the Internet of Things (IoT) include distributed components that need collaborate to achieve common goals. These systems are envisaged to provide essential services in areas including infrastructure inspection, smart city maintenance, manufacturing, and transportation. Furthermore, they need to be self-adaptive in order to cope with variability in workloads, changes in the environment, and changes affecting the system's components, such as a component leaving or joining the system.

The components of these distributed systems are typically required to perform tasks that are physically distributed, and inter-component communication is often affected by high-latency and low-bandwidth. As such, the control software of these systems cannot maintain up-to-date system-level models, and needs to be decentralised in order to respond to relevant changes that occur in the components and the environment by re-planning its actions accordingly.

In this thesis, we introduce nuDECIDE, a novel approach to achieving this decentralisation of the control software of distributed self-adaptive systems. nuDECIDE makes novel use of mathematical programming techniques to dynamically partition the goals of the self-adaptive system among its components, and probabilistic model checking techniques to ensure that each component achieves its sub-goals in the presence of environmental uncertainty. Three mathematical programming techniques are adapted for use within nuDECIDE, namely linear programming, integer programming, and policy synthesis for Markov decision processes.

Additionally, we introduce an application-independent software architecture and a reusable software platform (i.e., middleware) for developing self-adaptive systems with decentralised control. Finally, we present an engineering approach for using the nuDECIDE architecture and software platform to develop decentralised-control self-adaptive systems.

To illustrate the effectiveness of nuDECIDE, we used its goal partitioning techniques, architecture, software platform and engineering approach to develop decentralised software controllers for several distributed self-adaptive system from the CPS domain. We illustrate how the nuDECIDE components can be specialised for three case studies, which we developed using a combination of real mobile robots and simulation.

Table of Contents

List of Tables

List of Figures

Dedication and acknowledgements

I would like to extend my sincere thanks and gratitude to my supervisors, Dr. Radu Calinescu and Dr. Javier Cámara, for all the time and effort they have made throughout the past years in directing my research. Throughout my study, their guidance and comments enlightened me and helped me make this work a reality.

I would also like to thank all the Trustworthy Adaptive and Autonomous Systems group members for all the rich and meaningful seminars that had a profound impact on my research work. I am especially grateful to Dr. Simos Gerasimou and Dr. Colin Paterson for all the help and advice they provided me in overcoming many obstacles and challenges.

Special thanks goes to my colleagues that I have been fortunate to work alongside. In particular, I would also like to thank Naif Alasmari, Khaled Aldheeif, Ioannis Stefanakos, Misael Alpizar Santana, Emad Alharbi and Abdullah Albalawi.

I would also like to thank the University of Jeddah for the support I received throughout my studies, which contributed to making this project a reality.

Finally, I would like to thank my family for the ample support and encouragement I have received from them throughout my studies, making this project's realisation possible.

AUTHOR'S DECLARATION

Except where stated, all of the work contained in this thesis represents the original contribution of the author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

Parts of the research described in this thesis have been previously published in:

- Saud Yonbawi, Radu Calinescu. **Towards Self-Adaptive Systems with Hierarchical Decentralised Control**. In 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W). pages 14-16, 2018.

CHAPTER 1

INTRODUCTION

1.1 Self-Adaptive Systems with Decentralised Control

A self-adaptive system (SAS) is able to cope with uncertainty arises from changes in system's resources and environment. Research in SAS has received a growing interest in many computer science research communities. The research attention was on establishing the grounding principles and defining various architectures of adaptive systems[47]. Researchers paved the way to build systems capable of handling run-time changes without defaulting on system's constraints and objectives. A SAS interacts with changes in their resources and environment by adjusting system parameters or structure. The changes check is performed periodically to meet system functional and non-functional requirements [63]. A feedback control loop [68, 160]. A prominent control architecture comprises **M**onitor, **A**nalysis, **P**lan, and **E**xecute activities, which are usually referred to as the MAPE loop [105]. A knowledge repository can be augmented to the control architecture (MAPE-K) to reflect system state and requirements. The repository facilitates reasoning about system behaviour. In consequence, if the current behaviour deviate from system requirements, a set of corrective actions are planed to restore system confidence [92, 144].

There are many different applications that require the integration and collaboration of distributed self-adaptive components. These applications consist of interconnected components and can contribute to domains such as traffic management, advanced automobile, and infrastructure services (e.g., power, water and communications networks). Robotics is another domain that can provide services such as , telepresence, assisted living, and rescue operations [123]. Components of those application are loosely coupled and deployed on distributed hardware [169]. The nature of the tasks assigned to these software components requires the execution of parallel tasks distributed across a geographic area. This leads to the difficulty of having a prime component that is fully aware of the status of other components. The absence of global state renders rea-

soning about system behaviour in centralised manner a challenging task [34]. However, The absence of a leading component enhances the scalability of system components due to reducing the communication and computation burden [184]. In addition, tasks may require resilient to failures in execution, in which a system can recover from partial failure in components. This resilience may not be achievable when self-adaptation architecture is centralized or hierarchical. Therefore, the distribution of system resources and the nature assigned tasks play a pivotal role in the selection of the appropriate self-adaptation architecture.

1.2 Self-adaptive systems requiring decentralised control

This section illustrates class of problems that require the adoption of a decentralized control pattern in self-adaptive systems consisting of distributed components that closely coordinate to achieve common goals. System form domains such as such as Cyber-Physical Systems (CPS) and Internet of Things (IoT) often comprise distributed components that operate under high levels of dynamicity. These systems have to handle workload variation and sudden variability in system available resources in which components join and leave the system from time to time. Systems in such domains have to address uncertainty concerns raised by interacting with human users and operators, and dealing with changes in system environment and goals. Such system may serve in applications, which involve satisfying goals that are characterised as safety-critical. These goals often necessitate meeting strict functional and extra-functional requirements. For instance, a CPS that operates in domains such as healthcare and transportation has to adhere to strict functional requirements as the system interact with the physical environment. Similarly, the system has to satisfy extra-functional requirements that specify strict performance and dependability requirements. Accordingly, the control software of such system needs to address adaptation concerns and provide assurance that the system comply with these requirements.

While there are many research approaches that deal with various adaptation concerns and address the provision of guarantees in meeting the requirements of self-adaptive systems [47, 64], most of the effort addresses these issues in traditional software systems excluding systems comprise components that are physically distributed to perform their functions. The challenges in engineering the software control of these systems differ from those in traditional systems [17]. First, for systems that consist of heterogeneous components, which can be found in domains such as robotics, IoT and transportation, the engineering of the software control such that to provide guarantees in fulfilling their goals is a complex task. Consider for instance a multi-robot system comprises robots that vary in terms of their types i.e, mobile robots, drones,…etc, and sensing and actuating capabilities i.e, a drone has sensors and actuators that enable it to perform tasks that a mobile robot can not fulfill. Second, addressing adaptation concerns in a timely manner necessitates reliable communication between system components. Reliable communication is vital in order to maintain up-to-date global system models. However, having a reliable communication

i.e a one with low latency and high bandwidth to accommodate the high volume of message traffic, is not the case in many systems, that are often deployed with only affordable and unreliable communication between components, which is the case in systems from domains such as IoT and CPS. Third, even if reliable communication do exist, the analysis of such global models, which are often large, is inefficient and would impose delays in reacting to adaptation concerns. Last but not least, many software systems encompass components that belong to different organisations such that there is no trustworthy component that can oversee self-adaptation.

The decentralisation of software control is a key ingredient to tackle these challenges and pave the way to develop methods and software frameworks that contribute to expanding the use of self-adaptation in new domains. Decentralisation of control facilitates the application of self-adaptation to systems where the modeling and analysis at a system-level are impractical. Even if modeling and analysis are possible at the system-level, the approach as a whole would not be scalable, as it can not accommodate an increase in components. Also, if the system is required to meet safety-critical goals, it is unfeasible to use formal methods, which require an exhaustive analysis of the modeling space, as the models at a system-level are typically too large and their analysis would lead to common issues such as state explosion problem [53]. Thus, using a decentralised control resolves these challenges and yields an approach that is scalable and can provide assurance in fulfilling safety-critical goals.

For instance, consider the CPS of unmanned underwater vehicles from [34], the system relies on a decentralised control software to increase the efficiency in analysing mission-critical goals and allow the system to expand by adding more vehicles. Also, the decentralisation of control software increases the resilience of such a system to failures as the decision-making is localised and there is no single component oversees the self-adaptation decision. Another example, consider the IoT system from [181], the system can benefit from a decentralised software control such that to increase the scale of the system by adding more tenants. Such an increase in size is possible as the decentralisation of control relies on local decision-making and reduces the frequency of communications between tenants.

Nevertheless, decentralised control software is feasible when components of a system belong to different organizations and there are concerns (e.g., security or privacy concerns) in exchanging information required for the decision-making between components. For instance, a smart e-health system such as the one illustrated in [127] can benefit from the decentralised control software to localise the decision-making and avoid sharing patient sensitive information such as medical tests , diagnoses history,...etc. In summary, decentralising self-adaptation control enables dealing with heterogeneity and inherent distribution of system components. Also, the decentralisation enables localised decision-making, which in turn facilitates the use of formal methods to analyse relatively small models and provide assurance in fulfilling system goals.

Therefore, there is a growing research interest in using decentralized control to develop self-adaptive systems that include distributed components in domains such as cyber-physical

systems (CPS) and the Internet of Things (IoT),...etc. As the components in these domains require a high degree of autonomy, which can be achieved by localising the adaptation decision-making and coordinate when possible to meet system goals. The localised decision-making enables each component to deal locally with uncertainty arise from interacting with its environment when possible. [159]. Augmenting these distributed autonomous component with self-adaptive properties such as self-configuration, self-healing and self-organisation is an open challenge [155]. Tackling this challenge requires adopting a decentralized control in order to reduce the impact of limited communication and address adaptation concerns in timely manner [64]. There is a lack of software engineering approaches to decentralize adaptation control in distributed autonomous systems, while there is a large body of knowledge for engineering and experimenting self-adaptation in service based and traditional systems. For instance, there are many initiatives involve experimenting with self-adaptation in service-based applications such as TAS [178] and Rainbow [50]. Our aim is to expand the range of application domains that can employ self-adaptation to achieve system goals. To this end, our approach aims to support the development and experimentation of self-adaptation in Cyber-Physical application domain.

In particular, requirement characteristics in CPS applications differ from those of traditional systems. For example, consider a CPS, which consists of distributed components that are deployed in applications from domains such as smart transportation, smart cities, and ambient assisted living. Such system must adhere to strict functional and non-functional requirements as it interacts with the physical environment by sensing and actuating. Providing the necessary guarantees on satisfying system requirements require a continuous formal verification that all system components perform their assigned partition of the requirements. This verification is obtained at run time and is challenging to repeat continuously from a system-level after each change that affects one of the system components. System-level formal verification imposes restrictions on the size and complexity of the system. In addition, the nature of tasks in such systems requires a decentralised self-adaptive control since the components of these systems usually partition spatial tasks to meet their functional requirements. The decentralisation of self-adaptive control allows these components to cooperate to fulfill system functional requirements by partitioning these tasks and assign each to a system component. Nevertheless, the decentralised control enables the components to deal with restrictions of communication, and enhance the resilience of the system against failures in any of its components.

1.3 Formal Methods

Formal methods are mathematically based techniques supporting the rigorous development of computer-based systems, including software and hardware systems. The approach considers the specifications, requirements and design of such software. It provides the means for developing and programming software systems that require robustness and rigidity in executing assigned

tasks [54]. Applying this approach to manage the behaviour of self-adaptive systems is an active research area [38]. Runtime quantitative verification (RQV) is a promising technique that utilises stochastic models to represent the behaviour of a SAS and its environment. It provides the necessary tools to verify system requirements at runtime. These requirements are expressed as a temporal logic formulae and verified continuously in a closed-loop control. However, the effectiveness of this technique depends on the size and structure of the verified model. The larger and more complex the model is, the more time is required to verify system requirements. This issue is due to the exponential growth of model size as a result of expanding the size of the system. This problem is commonly known as state explosion problem [11]. Thus, the increase of system size reduces the effectiveness of applying RQV techniques in self-adaptation [39, 70, 73]. This may cause computation and memory overheads. The large system size introduces delays in reacting to adaptation concerns. Therefore, this research explores the probabilistic model checking [61] variant approach to reduce the time and computation required in the runtime verification. In addition, it adopts the use of a decentralised adaptation control to reduce the number of the verified states [184]. A decentralised control approach as [34] manages to reduce computation and memory overheads by dividing the burden between distributed and coordinating MAPE control loops.

Mathematical Programming (MP) can be employed as a type of formal methods used for solving a broad range of problems found in domains such as finance, marketing, and personnel management. MP provides a set of alternative mathematical models that captures specific domain problems. Particularly problems that require optimisation to find the best solution. The mathematical model comprises a decision variables, set of constraints, and an objective function. The decision variables describes the a possible solution (x_0) and define the values a variable can hold. A possible solution x_0 has to satisfy equalities or inequalities constraints to be a candidate solution. The relationships between the decision variables and applied set of constraints defines the potential solution space (A). The quality of the candidate solution x is measured using the objective function $f(x)$. An optimal solution is a feasible solution that maximises or minimises the objective function. Thus, an optimal solution can be found given the following function

$$f : A \rightarrow \mathbb{R} \tag{1.1}$$

The element $x_0 \in A$ is considered an optimal solution if value of its objective function $f(x_0)$ is maximum (or minimum if the goal is to minimise) for all $x \in A$ such that $f(x_0) \geq f(x)$. since our approach favors a decentralised adaptation control, a variant of mathematical programming model is required to plan an optimal configuration for partitioning system tasks. The global allocation of tasks require a deterministic optimization technique that yields an optimal configuration. MP is a candidate technique for efficiently solving this class of problems.

Another formal approach that supports decision making in a wide range of applications is policy synthesis for Markov decision processes. Markov decision processes (MDPs) [16] are discrete-time stochastic processes used in decision making in applications with optimisation

problems. MDP enables decision-making in such applications by providing a mathematical framework for optimising the chosen decisions' outcome. In an MDP, a decision problem comprises several states, where each state has some actions that may lead to multiple random outcomes. Typically, a decision-maker can control which action to follow from each state by associating the decision-outcome with quantitative rewards. An MDP policy specifies which action to choose from each state such that the non-determinism in an MDP model is solved. In particular, solving an MDP entails synthesising a policy that defines which action to select at every state of the MDP. The selection of actions is based on some property that needs to be achieved (e.g., maximising/minimising some rewards structures defined using a real function). There are a plethora of algorithms that are implemented as libraries and software tools, which can be used to solve MDP. For instance, there are many probabilistic model checking engines and planners [65, 119] that employ algorithms based on techniques such as linear programming and policy iteration.

1.4 Motivation and Research Hypothesis

In recent years, considerable research effort has been directed towards adding self-adaptive capabilities to software controlled systems in multiple application domains [105, 138]. This involves augmenting systems with a feedback control loop [68, 160], typically comprising monitor, analyze, plan and execute (MAPE) steps, and a knowledge repository that captures the system state. The feedback mechanism allows the analysis of the current system behaviour (reflected in the knowledge repository) against the predefined requirements and intervenes when a deviation is detected. In consequence, a set of corrective actions are planned and eventually executed to continue to meet the system requirements [92, 144]. The proliferation of affordable cyber-physical components has led to a growing interest to develop Cyber-Physical systems (CPS) that employ the synergy and integration among their components to achieve common objectives. The physical and computational components are intertwined and operate in a different spatial and temporal scales. A CPS component consists of software on an embedded computer that implements one or multiple feedback loops to monitor and control the physical processes. The changes in the physical processes and execution context affects the computations in the software component and vice versa [124].

Recently published research roadmaps identify the open challenges in engineering self-adaptation in distributed application [47, 64]. The first challenge is the effective decentralisation of the adaptation control to mitigate the risk in centralising the computation and communication by assigning all the load to a single controller. The decentralisation of adaptive control eliminates the single point of failure and improves the potential in scaling the controlled system size. This control pattern accommodates the nature of distributed systems, in which no superior component oversees and controls peer components. The decentralised adaptive control pattern is

more resilient against changes in network topology, where a remote peer may fail unexpectedly. Therefore, applying this control pattern on distributed systems increases flexibility and conforms with the long-term autonomic computing vision [105].

The second challenge arises from safety-critical domains, which require strict compliance with requirements in uncertain operating environments [31, 63]. Applying formal verification at system level limits the size and complexity of distributed CPSs for which this compliance can be achieved, especially if the verification needs to be performed at runtime. Weyns et al. [184] describe patterns for decentralising MAPE loops, e.g. to efficiently use verification techniques at runtime [180]. A distributed self-adaptive approach has to adjust its collective behaviour to retain fulfillment of quality requirements (e.g, performance, efficiency, and reliability), while also using an optimal or near-optimal configuration that satisfies global requirements without undermining the component's local requirements. Formal methods provide the mathematical foundation to formally model system characteristics, but their application to distributed self-adaptive systems is underexplored.

The hypothesis pursued by the research presented in this thesis builds on a distributed SAS control decentralisation philosophy introduced in [34]. Called DECIDE, this philosophy proposes that:

1. the components of a distributed SAS operate independently of each other at most times, only exchanging the summary information required to partition (or re-partition) the SAS goals among them when the system is deployed, and after "major disruptions" (e.g., when a robot sensor or actuator fails);

2. both the component-level control and the system goal partitioning are carried out using formal techniques capable of providing guarantees that the system goals will be achieved.

The preliminary evaluation of this philosophy provided in [34] shows its benefits for a small team of homogeneous unmanned underwater vehicles that perform a simple monitoring mission. The research work described in the thesis sets out to significantly extend and simplify the applicability of this result, under the hypothesis that:

1. New ways of exploiting mathematical programming techniques can be used to ensure that DECIDE-style goal partitioning can be carried out within distributed SAS with (a) heterogeneous components, and (b) for a broad variety of distributed SAS application goals;

2. The DECIDE principles can be captured within an application-independent software architecture, a reusable software platform (i.e., middleware) that reifies this architecture, and an engineering approach for specialising this platforms for a distributed SAS under development.

1.5 Thesis Contributions

The thesis advances the development of distributed self-adaptive systems with decentralised control with the following contributions.

1. A repertoire of techniques for partitioning the goals of distributed SAS among their components. We introduce three new methods for partitioning distributed SAS goals based on the contributions that the SAS components can make towards achieving these goals. The three methods employ different types of mathematical programming techniques, i.e., linear programming, integer programming and policy synthesis for Markov decision processes. Each of these methods can handle a particular class of system goals, so they are complementary to each other, as explained later in the thesis.

2. The nuDECIDE application-independent software architecture for decentralising the control of distributed SAS. We introduce a hierarchical-control software architecture to support self-adaptation in distributed SAS. The new architecture comprises a system-level control loop that partitions the goals of the distributed SAS among its components under conservative assumptions and component-level control loops responsible for achieving the component sub-goals. The operation of the two control loops is underpinned by models that are small enough to enable the use of formal analysis techniques. The architecture enables the provision of assurances for distributed SAS deployed in applications that cannot be easily handled by existing approaches.

3. The nuDECIDE reusable software platform (i.e., middleware) that reifies our sofware architecture, and that can be specialised to support the development of distributed SAS with decentralised control. The software platform comprises reusable software components that implements nuDECIDE common functionality as a library of application-independent software components. This software platform is specialised to devise the decentralised controller for three case studies (i.e., the case studies are introduced in Chapter 3) which illustrate the effectiveness of the platform for instrumenting several distributed SAS from the CPS domain. This software platform extends the preliminary work in [34] in which the software platform used to devise the decentralised control is tightly coupled with an application from the marine monitoring domain.

4. An engineering approach for specialising the nuDECIDE software platform for a given distributed SAS. We introduce the four-step approach for using the nuDECIDE architecture and software platform to develop the decentralised software controllers for a distributed SASs. These steps are essential to devise the decentralised software control for a distributed SAS. The description of the steps is accompanied with a running example that illustrate the use of the engineering approach in an application from the infrastructure inspection domain. The first step comprises devising the probabilistic models that captures the behaviour of the components of a

distributed SAS. In the second step, we identify the parameters that influence the behaviour of the devised models. The third step involves describing how the component-level and system-level requirements are specified. Finally, the last step describes the development process aimed at specialising the reusable software components of nuDECIDE software platform to devise the decentralised control software for a distributed SAS.

5. An extensive evaluation of nuDECIDE through experiments using both real mobile robots and simulation. We introduce three case studies used to evaluate the nuDECIDE approach, including its different system-goal partitioning techniques and reusable software platform. The first case study comprises a team of three mobile robots that simulate carrying out a pipeline inspection mission in a lab environment. The case study includes the use of the linear programming method to partition the pipeline route, that requires the inspection, among these robots. The second case study involves simulating the execution of a waste management mission using a team of mobile robots. The mission comprises deploying these robot to collect both garbage and recyclables from multiple locations with a public park. The system goals in this case study are encoded as as an integer programming problem and solved accordingly. The last case study also simulates the execution of a waste management mission using a team of mobile robots. however, this case study includes the use of two variants of the MDP policy synthesis method in partitioning system goals. Unlike the linear and integer programming methods, the MDP method can consider the probabilistic behavior of system components when partitioning system goals.

1.6 Thesis Structure

The remainder of the thesis is structured as follows.

Chapter 2 introduces the concepts, terminology and modelling paradigms used by nuDECIDE. This includes describing the architecture of a decentralised control loop and illustrates the various interaction patterns in this architecture. As for the modelling paradigms, we introduce discrete-time and continuous time Markov models and Markov decision process. This chapter introduces variants of probabilistic temporal logics, which are used to specify properties for these models. The chapter as well describes the DECIDE approach that underpin the research carried out by the PhD project. Finally, overview a number of representative approaches to decentralising the control software of distributed self-adaptive system.

Chapter 3 presents the repertoire of nuDECIDE goal partitioning techniques that constitutes the first contribution mentioned in Section 1.5. For each technique, we first introduce its theoretical foundation, and then provide a detailed example that illustrates its application.

Chapter 4 describes the nuDECIDE framework for the engineering of the decentralised control software of self-adaptive systems. The framework comprises a decentralised control architecture (presented in Section 4.1), a reusable software platform (i.e., middleware) that

uses the goal partitioning techniques from Chapter 3 and reifies the architecture (presented in Section 4.2), and an approach for specialising the components of the software platform for a specific application (detailed in Section 4.3).

Chapter 5 describes the extensive experiments carried out to evaluate the goal partitioning methods introduced in Chapter 3 and the nuDECIDE framework introduced in Chapter 4. We performed this evaluation for a range of case studies, using both real mobile robots (for the linear programming method from Section 3.3, Chapter 3) and simulation (for the integer programming and MDP strategy synthesis techniques from Sections 3.4 and 3.5 in Chapter 3).

Finally, Chapter 6 summarises the contributions of the thesis, discusses their limitations, and identifies directions for further research on decentralising the control software of distributed self-adaptive systems.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1 Self-Adaptive Systems

Weyns [177] describes the common interpretations of the concept of self-adaptation and highlighted the two most common definitions of what constitutes a self-adaptive system (SAS). The first interpretation stems from the mechanism used to realise self-adaptation. A self-adaptive behaviour is realised by using a closed feedback loop to monitor and adapt the system's behaviour at runtime. The first interpretation focuses on *external* mechanisms that enable the system to address adaptation concerns. The definition disregards traditional *internal* techniques such as using exceptions clauses in programming languages and error-tolerant protocols to deal with events that necessitate self-adaptation. The use of external techniques divides the SAS into a part that deals with domain concerns related to the goals for which the system is built. Another part deals with adaptation concerns by looking at how the system realises its goals under changing circumstances. This division of concerns promotes the "disciplined split" view of how the SAS is conceived.

The second interpretation for a SAS considers the existence of uncertainty in the environment or the domain in which the system is deployed. The definition views the SAS as a system capable of controlling its behaviour in response to the perception of changes in the environment and the system itself. According to this interpretation, a system that has been improved with self-adaptative capabilities is seen as a black box affected by dealing with changes imposed on the system when interacting with external conditions such as dealing with failures, change in the workloads, availability of resources, and demands. The self-adaptive capabilities enable the system to deal with these changes by using the means provided by the system to address adaptation concerns.

A system that exhibits self-adaptive characteristics has the ability to adjust their structure

and behaviour to attain these requirements. The stakeholders of such a system can specify a set of objectives to accomplish certain tasks under predefined QoS restrictions. The are many factors that increase the complexity of today's software system. Factors such as the variations in the availability of system resources and the heterogeneity of system components raise the complexity to the extent that managing them using human operators is in many cases, simply not feasible. Self-adaptation provides the means for such systems to realise their goals under changing circumstances, thus helping them address these challenges.

For example, a system may experience variability in resource availability, which can be tackled through self-adaptation to increase system resilience when a system resource fails to fulfil its goals. In recent decades, self-adaptive systems (SAS) have received considerable interest from researchers within multiple areas of computer science. In particular, research on establishing grounding principles and architectures of adaptive systems has gained momentum. A review of the literature that has been published in this space over the last decade indicates a relatively recent increase in the research effort in the domain of SAS from a software engineering perspective [29].

The rapid expansion of distributed pervasive computing systems in recent years has led to an unprecedented level of complexity in development of SASs. Such systems need more frequent updates as they deal with changes at runtime in their requirements and operating environment. Changes render those systems unable to achieve their goals and makes them vulnerable to errors. Thus, traditional approaches to developing such systems are unfeasible in meeting systems goals [163]. The problem exacerbates in systems comprised of distributed components that cooperate to achieve the goals of the system. Each component interacts with an unstable environment that affects the system's stability as a whole and makes it vulnerable to failure. The development of such systems in a way that makes them stable and able to deal with the dynamic changes that affect their distributed components is a complex task due to several factors shown in Figure 2.1.

In the past, the majority of the research effort in software engineering has concentrated on handling complexity and achieving quality goals during design time such as in ISO 9126-1 quality model [76]. In contrast, self-adaptive systems substitute the traditional software development cycle methods with approaches that are focused on system evolution during runtime with less human operator intervention. The introduction of self adaptation in software systems is based on the incorporation of feedback control loops [29]. The feedback control loop permits alteration of system behaviour to address adaptation concerns. Precisely, self adaptation is carried out by modifying the system structure or behaviour to address requirement violations. This in turn leads to changes that may occur at the system boundary or its environment. Consider, for example, a service-based system comprising several components. A degradation in the performance of a system component may lead to delays in response time. The delays can be countered at runtime by adjusting the system's behaviour (e,g, using another component with better performance to deliver the service).

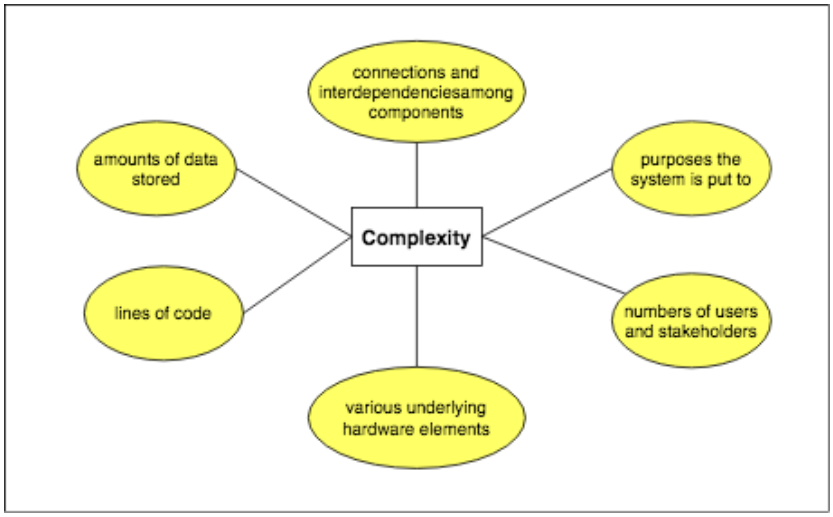

FIGURE 2.1. Origin of complexity in computer systems.

Being able to recover from undesirable situations at runtime, as well as optimizing some QoS metrics such as cost, performance, and reliability requires architectural support since developing these systems to make them self-adaptable requires looking at the mechanisms that enable representing the behaviour of such systems and reasoning about their behaviour to check if they fulfil their goals. In order to realize that vision, the notion of self-adaptive system architecture has been explored by different communities. Therefore, it is essential to formulate a definition that sufficiently covers a wider range of the spectrum. A recent published book assimilates the different interpretations and proposes a unifying definition [126]. They identify self-adaptive systems as software applications that constantly preserve an active representation that models the system and its environment. The preserved representation allows the recurrent reflection of modifications that influence the system requirements. Consequently, a knowledge repository is maintained to accommodate the accumulated knowledge and to reason about system behaviour. Then, if the current behaviour deviates from the system requirements, corrective actions are planned to revert to system behaviours that satisfactorily meet the requirements. Figure 2.2 illustrates the recurrent accumulative knowledge through feedback mechanism. Also, the Figure depicts the utilisation of gained knowledge to analyse and reason about system behaviour. The continuous analysis and reasoning retains a persistent awareness of any violations of system abstract higher-level aims. System aims are defined to clarify system functionality and the scope of optimization. In order to accomplish that, self and context awareness must be enforced periodically to reason and act accordingly [92, 144]. However, there is no unified model that captures and represents the changes in software states. Engineers can refer to any feasible form to reflect the current state of underlying system.

Another aspect to consider in the self-adaptation architecture is how Knowledge is modeled. Representation models are divided based on their envisioned role into three main categories.

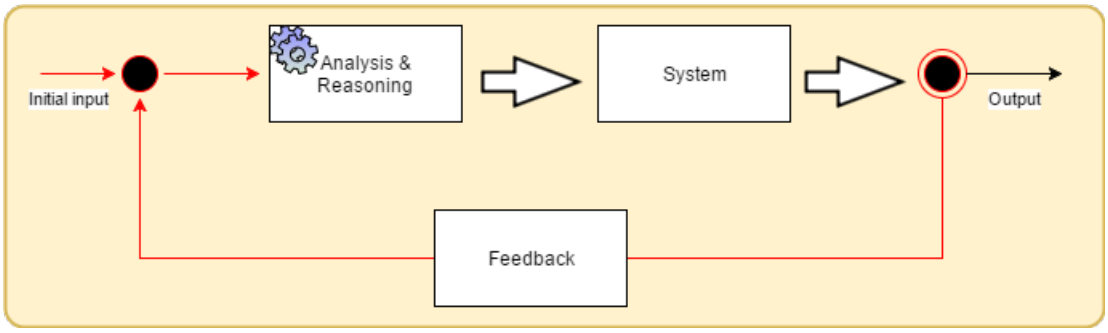

FIGURE 2.2. Feedback control system.

The first category represents system abstract goals that coexist with required relevant metrics. For instance, a descriptive knowledge representation deal with the appointed system objectives and the governing policies. The second category employs a predictive modelling approach to support the forecast of an adaptive decision outcome or to illustrate the execution pattern under certain constraints. The prediction is realized through exploiting the causal relationship between input signal and required output behaviour. An example for this model is in systems that exhibit stochastic behaviour pattern [39]. The prediction model can foresee system throughput, in exchanged to utilised resources and cost. The final modelling paradigm concentrates on organizing conceived action plans under a given incident.

A survey published in 2009 advocates the organization of self-adaptive properties in a hierarchical view [155]. As depicted in Figure 2.3, the top most layer illustrates self-adaptation as the general aim. Followed by another layer that constitutes four different properties (Self-Configuration, Healing, Optimization, and Protecting) for driving self-adaptation. A description of these four themes which are reported in Kephart and Chess's [105] seminal work, is as follow:

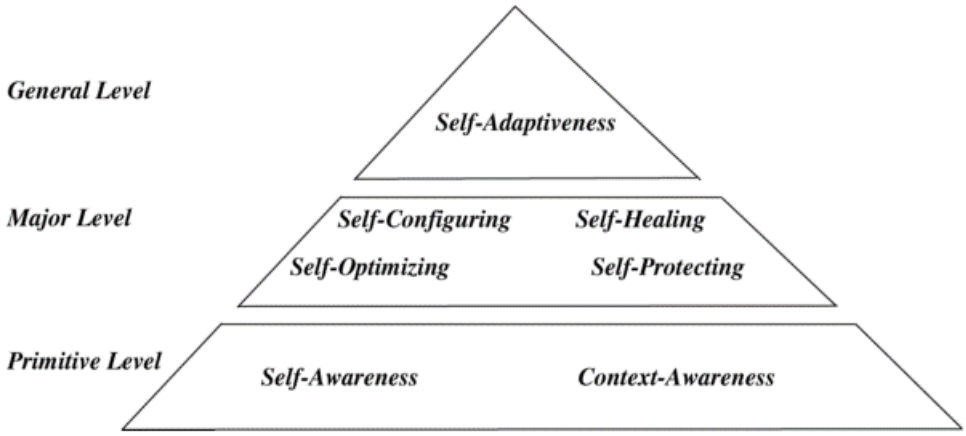

FIGURE 2.3. Hierarchy of the self-* properties [155].

- **Self-configuration.** In a dynamic context, an autonomous system must reconfigure in response to internal and external variations. For instance, different functionalities are obtained from composing components in interchangeable manner with a view to derive certain behaviour
- **Self-repairing.** Self-healing implies the ability to locate failures by self-diagnosis and to recover in a reactive or proactive manner. The self-healing property allows recovering systems that may be affected by both partial and complete failure patterns [62, 151].
- **Self-optimization.** This property focuses on satisfying several non-functional (QoS) metrics and optimising some of these metrics by adjusting predefined controller parameters. For example, a service-based system may be required to fulfil requirements related to efficiency, performance and reliability while at the same time improving response time (performance) when the system provides services [165].
- **Self-protection.** Similar to self-healing, self-protecting systems are able to distinguish and respond to security threats before or after occurrence [192].

The foundation layer describes self-awareness and context-awareness properties, which are essential for fulfilling any of the attributes on the other layers [95, 154]. The former layered organization of self-* attributes is advocated and widely accepted by the self-adaptive systems research community, as confirmed at several Dagstuhl seminars [47, 64]. This research group provides roadmaps for all aspects of engineering self-adaptive systems, particularly concentrating on aspects such as decentralization of control and assurance. Over the past decade, there has been significant effort in recognizing self-adaptation across multiple domains, which eventually contributed in diversifying concepts and interpretations of methods for engineering a self-adaptive system. Hence, the analysis and design of extensible engineering methods is crucial to address present deficiencies that restrain the broad acceptance of adaptation [29, 155]

2.1.1 Taxonomy of Self-Adaptation

Krupitzer et al. [114] describe the five important concerns that have to be addressed prior to the implementation of SASs. The five concerns are depicted in Figure 2.4, and examined in subsequent sections.

2.1.1.1 Time

The temporal concern of self-adaptation is essential for understanding the triggering mechanism of any self-adaptive action. It is customary to initiate adaptive plan in a reactive manner, i.e, recovering the underlying system from faulty state or resolving a performance issue after its occurrence. A proactive style on the other hand, enforces a predictive method to foresee any disturbance before its appearance [114]. Although, a proactive adaptation is more convenient

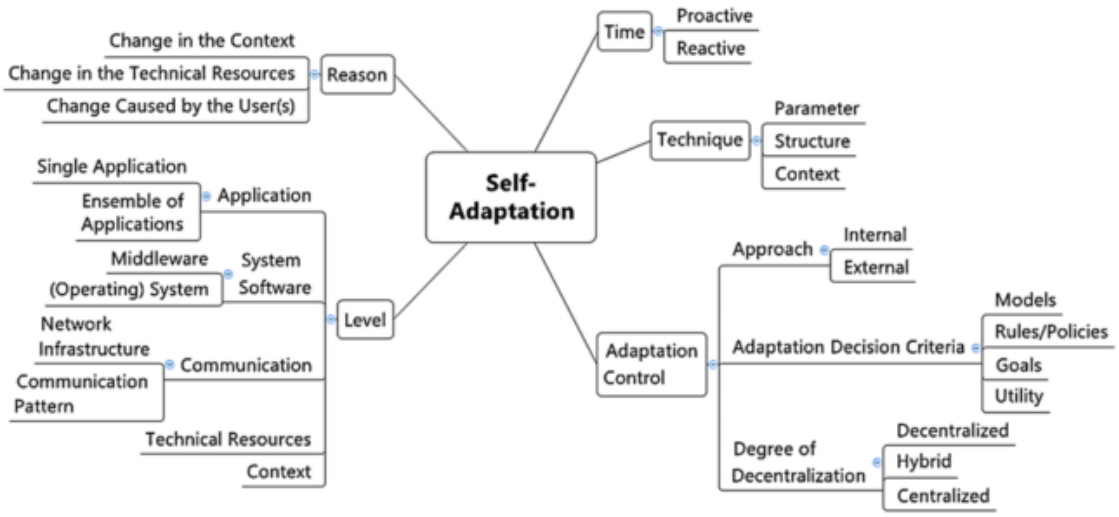

FIGURE 2.4. Taxonomy of self-adaptation requirements [114].

than a reactive one as the former avoids unnecessary interruption in system schedule. There is a considerable risk in embracing the proactive temporal, which may occur as a result of incorporating faulty predictive approach. Unlike a reactive approach, implementing a proactive approach is a non-trivial task. Hence, a reactive adaptation decreases the risk of false positives and avoids adversarial adaptation. Proactive approaches, on the other hand, have been shown to yield closer to optimal solutions, compared to reactive adaptation [114].

2.1.1.2 Reason

As discussed in Section 2.1, a SAS needs to reason about its behaviour and plan for adaptive actions when the behaviour of the system violates its requirements. To synthesise an adaptation plan, a SAS examines all possible corrective actions and executes the most suitable one. In general the reasons for triggering an adaptation action can be any of the following described below:

1. a variation in the system itself, for instance a defect in a system component, or an optimization of workload among components, may trigger an adaptive action;

2. a variation in system environment as for example a server experiencing a burst on requests, which require processing in accordance to predefined timing restrictions;

3. a modification required by users, in case of changes in the set of system objectives that yield adaptive behaviour.

Therefore, a continuous observation of system interactions is required to detect the reasons for triggering an adaptive decision [114].

2.1.1.3 Level

In an article published in 2013 [184], Weyns et al. emphasised the principle of separation of concerns [129] for dividing SAS into two essential parts. These parts are the managed and managing subsystems. The view of separation of concerns is widely advocated by researchers in the area of SAS [182]. A SAS has a managed sub-system, which is directed by a managing sub-system that incorporates the adaptation logic. The managed sub-system provides the means that enable the adaptation logic from perceiving the change in the system's behaviour and enables the adaptation logic from applying the adaptive actions if the behaviour violates the system's requirements [1]. For instance, consider the unmanned marine vehicle from [34], which has to survey a certain distance under variations in system resources. The managing part in this scenario has to monitor the available system variables such as, movement speed, sensors reading accuracy, and energy utilisation, to adapt when necessary in order to complete the required task. Another example is found in the Rainbow framework [79] where the managing system partition is a self-adaptive layer that instantiates the framework and incorporates a set of architecture-centric models updated at runtime, whereas the managed system partition is the system layer.

Nevertheless, an SAS operates within an environment that affects its functional and non-functional behaviour by means of interactions. In general, a self-adaptive system observes the changes in its environment [100]. For instance, the environment of an SAS can be comprised of network elements with variable traffic load, hardware components, other remote systems that interact with the SAS, and even physical objects. Consider the earlier unmanned marine vehicle example where the vehicle encounters physical obstacles such as rocks during its operation. In such a scenario, the vehicle has to avoid collision with the rocks by adapting its path. The previous example indicates the inherent limitation of SAS to control their environment. Figure 2.5 summarises the relationship between the managing and the managed subsystems while also illustrating the relationship of the SAS with its environment.

2.1.1.4 Adaptation Techniques

Krupitzer et al. [114] combine and refine the views of Handte [90] and McKinley [132] on adaptation techniques, classifying the techniques into three main categories. These categories are: (i) structure, (ii) parameter, and (iii) environment adaptation techniques. The structure adaptation technique yields changes in the structure or the functional role of components in the adaptive system. For example, the structure change refers to the introduction/removal of components from the system or to the exchange of functional roles between components. Furthermore, changes may also occur in the composition links between components to constitute adaptive behaviour.

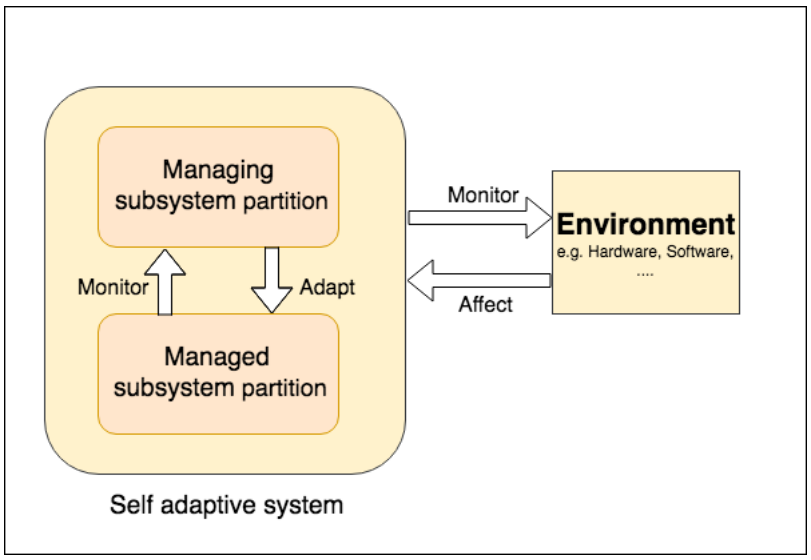

FIGURE 2.5. Self-adaptive system interactions inspired from [121]

The system component could be a variety of software and hardware resources, or even the system contexts, that could be considered as structure adaptation. The second category of techniques is called parameter adaptation. Adaptive behaviour is delivered through modifications in system parameters. However, adjustment of system parameters might be a non-trivial task if there are dependability relations between theses parameters. The last technique is environment adaptation, which Krupitzer [114] argues is something that can be often found exclusively in the area of pervasive computing, and not elsewhere. A pervasive adaptive system can sometimes alter environment variables given an adequate actuator. Eventually, it is conceivable to combine multiple adaptation techniques to derive effective self-adaptive behaviour.

2.1.1.5 Adaptation Control

There are various aspects to designing the control logic of a SAS. Adaptation control logic can follow the separation of concern principle by splitting the managing and managed subsystems using a modular design pattern [75, 155]. Alternatively, the adaptation logic can be intertwined with or embedded in other components of the managed system. Following separation of concerns increases the reusability of the adaptation control and also improves the maintainability of adaptive systems.

A second aspect is concerned with the extent to which adaptation control is centralised or decentralised. The centralised approach is feasible in a system with limited components, while the decentralised approach enables increasing the size of the managed components. Weyns [184] describes the benefits of decentralised control in certain scenarios and argues that a better scalability in computation and communication can be achieved in a SAS with decentralised

control. Mainly the decentralisation of control suits distributed managed components, in which no component can be accountable for handling the adaptation decisions. In addition to the centralised/decentralised approach, there is a hybrid approach that combines both of them. For instance, a *"parent"* central control component manages two decentralised siblings that coordinate with each other. The topic of distributed self-adaptive system with decentralised control elaborated in more detail in Section 2.2.2.

The final aspect to consider is the knowledge representation of a target managed system which is vital for reasoning and deriving the appropriate adaptive behaviour. Lalanda et al. [121] state that knowledge representation plays a vital role in adapting managed systems. It also makes reasoning about the system less complex and easier to deduce required adaptive behaviour. In fact knowledge representation can facilitate the provision of measuring metrics to support an adaptation decision against alternatives. Knowledge representation types are divided as follows:

1. **Rule-based structure -** Commonly known for its simplicity and provisioning straightforward reflexive adaptive action following the structure of event-condition-action (ECA). However, a careful examination of the utilised set of rules is essential to eliminate any conflict of events, which may result in adverse actions.

2. **Model-based structure -** Models capture the states of the managed system and its environment to predict and analyse the next adaptive actions. It employs trend analysis to reason about and devise adaptive behaviour. Models are constructed to ease the resolution of problems.

3. **Goal-based structure -** Goal based structure envisages the desired target state of system artefacts, allowing the adaptation logic to deduce adaptive actions.

4. **Utility-based structure -** Utility structure is used to resolve conflict in the previous goal structure by providing necessary measurement of usefulness to rank required goals.

To conclude, knowledge representation is an important aspect to consider in designing adaptation logic. A combination of knowledge structure can facilitate the adaptive decision-making and ensure that the pursued decision is optimal.

2.1.2 Engineering Self-Adaptive Systems

Feedback control is a key element in introducing self-adaptive capabilities in software systems [68, 105, 160]. Kephart and Chess [105] were the first to introduce the concept of an autonomic manager in software engineering. As shown in Figure 2.6, the automatic manager interacts with a managed system through two main interfaces. The first interface is known as sensors, which facilitate the measurement of system‘s attributes during the monitor phase. These attributes capture the system‘s current state and provide the essential metrics to evaluate system satisfaction of functional and non-functional requirements to the analyse phase. The analysis phase

in return is followed by the planning phase, which responds to violations in requirements by arranging a set of corrective actions to align the managed system with the necessary requirements. Thereafter, these actions are performed by the execute phase utilising a second type of interface called actuator, which adapts the underlying managed system [26]. These components share information using a common knowledge repository. The combination of the five components is often termed a (MAPE-K) loop in the research literature. Accordingly, engineering an SAS has long been perceived to require the development of these five components of the MAPE-K loop.

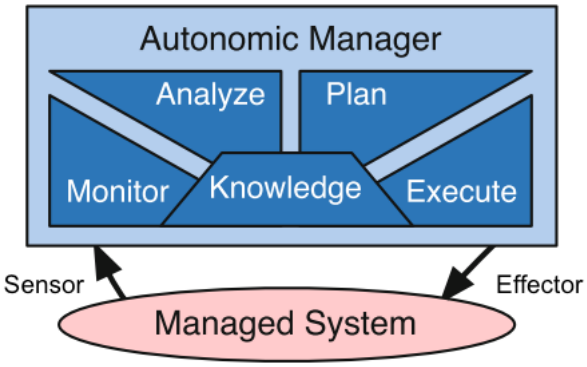

FIGURE 2.6. The MAPE-K model [26].

As shown in Figure 2.6 the autonomic manager, which holds these components, interacts with a managed system through sensors and effectors. These sensors and effectors interface with the managed system and expose its state (sensing) to allow the management of its resources (effecting). IBM's definition of a blueprint architecture for autonomic computing [1] refers to these interfaces as *touchpoints*, which implements the necessary functions to sense the state of the managed system and act on the adaptation decisions through the effectors.

Similarly, the SAS architecture proposed in [3] and depicted in Figure 2.7 promotes the split of the autonomic manager from the managed system. The main concern of the managed system is to derive domain functionality through the implementation of domain logic. This Allows the managing system, which implements the adaptation logic, to meet its essential role to autonomously adapt the behaviour of the managed system. A feedback controller following monitor-analyze-plan-execute (MAPE) architecture [105] is embedded in the adaptation logic which enables adaptation of the managed system. Ultimately the SAS executes in an environment which cannot be controlled and subsequently adapted, but, can be observed by the adaptation logic [26]. Another example is the hierarchical architecture suggested in Kephart et al. [105] for organizing the adaptation logic in top-down format where the parent controller orchestrates adaptation decision.

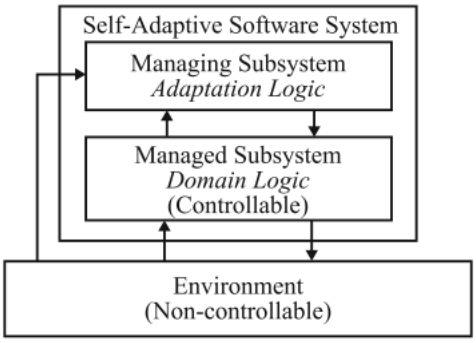

FIGURE 2.7. An architecture for Self-Adaptive Software Systems [3].

2.2 Distributed Self-Adaptive Systems

2.2.1 Description

Many important software systems consist of distributed interconnected components that are deployed on different hardware. Tel [169] defined a distributed system as a group of loosely coupled hardware or software artefacts that are connected via some type of network. According to this definition, artefacts have to work in isolation of each other and infrequently exchange information to achieve overall system objectives. Distribution can occur at the data, service or algorithmic levels [168]. The best example of distributed systems is found in web applications. There is an unprecedented demand for using the Internet of distributed applications that are provided worldwide. Fitzgerald describes the shortcomings in handling the staggering increase in produced data and the lack of available software systems to utilise this data as the new software crisis [74]. Google estimated the available internet data was five thousand petabytes (5 million terabytes) in 2005, and barely 0.004% was handled by their firm. They also predicted an exponential increase of the available data that could potentially double in 2010 [157]. Likewise at the hardware end, there is an exponential growth in the computation capacity which doubles each year and half according to Moore‘s Law [156]. In 2010 a then leading futurist at Cisco released an estimation of the number of devices connected to the internet. According to their report, there are approximately 35 billion devices that utilise the internet. That is, around six devices for each person who was living in 2010 [71]. This report also estimated that the number of devices will continue to grow and could potentially reach 100 billion connected devices by 2020. This exponential pattern of increase in data and computation power indicates the intrinsic need for distributed systems [74]. Therefore, distributed applications such as multi-agent systems, internet of things (IoT), and pervasive computing and robot swarms, have been widely embraced to exploit the enormous size of the data and computation capacity [6, 72, 152, 186].

These applications require better reliability and performance to function in accordance to their

requirements. Application requirements in turn are achieved by incorporating self adaptation techniques [46]. Adding self-adaptation to existing distributed software systems extends its capabilities to leverage on the obtainable computational capacity and data.

Distributed self-adaptation can be realised through centralised or decentralised control schemes. Work by Lemos et al. [64] has focused on two factors for recognising suitable schemes for the developed system. These factors are system characteristics and adaptation requirements. The system characteristics such as the architecture and size of a system, at hand could impose restrictions on deciding the control scheme. Therefore, adopting a centralised approach is more feasible when a single node that preserves the system‘s global information exist. In contrast, if the information required for adaptation is distributed across system nodes and there is a limitation in preserving the global knowledge, a decentralised control scheme is more appropriate. A decentralised scheme is essential when the system's architecture is constantly changing, in which no component can be accountable for handling the adaptation decisions. A component-based system, for instance, reflects the need for decentralised control for adopting an open system in which components enter and leave constantly [30].

The last factor to consider for determining the appropriate control scheme refers to the requirements of adaptation. In a centralised approach, fulfilment of the system‘s global requirements is a trivial task in comparison with decentralised control in which local requirements are prioritized. The satisfaction of the system‘s global requirements in decentralised control is complicated and relies heavily on constant coordination between system constituents. A promising approach to reduce the coordination effort in distributed self-adaptive systems with decentralised control is evident in the work of Calinescu et al. who managed to decrease the effort of coordination by establishing a contribution-level agreement (CLA) among the system components [34]. The initiation of contribution-level agreement is infrequent and occurs when components join or leave the system. Another area where decentralised control surpasses the centralised control is scalability. Centralised approaches place constrains on the number of distributed components. It is both difficult and infeasible to retain a consistent view of managed components in large distributed systems [23, 142]. Directing large number of partitioned components with a single control manager causes a considerable delay in reaction to changes in the components. This delay is a result of the concentrated direction of the adaptation-related information to a single control manager. Following that, the controller has to analyse the perceived information and adapt when appropriate. Georgiadis et al. [82] highlight how efficient scalability can be obtained with decentralised control in distributed systems. Such efficiency is achieved by excluding a single consistent view of the system configuration. Instead, each local manager concentrates on its managed resources and coordinates when possible with peer managers to deliver system wide adaptation requirements. Thus, the decentralised approach eliminates any single point failure by distributing the adaptation decision across multiple loosely coupled coordinating and controlling entities. Table 2.1 elaborates on differences between centralised and decentralised approaches.

Table 2.1: Capabilities of centralised and decentralised approaches

Control approach	System characteristic		Adaptation requirement		
	Dynamic reorganization of system architecture	Preserve system global state	Reliability	Responsiveness	Scalability
Centralised control	Partial supported [a]	Supported	Partial supported [a]	depends on managed system size	Partial support
Decentralised control	Supported	Not supported [b]	Supported	Prioritising local adaptation	Supported

[a] Support is restricted for changes that exclude system accountable control manager.
[b] System global state is distributed across system components.

2.2.2 Classification of Distributed Self-Adaptive System Control

There is a growing interest in the design of decentralised MAPE loops to adapt distributed systems [27, 69, 82, 130, 174, 183]. Weyns et al. [184] argue that instituting self-adaptation in a complex and heterogeneous software environment demands the employment of multiple coordinating MAPE loops. They also advocate further exploration of effective techniques in implementing distributed self-adaptive systems with decentralised control. There are various interaction patterns for organizing MAPE components in a decentralised manner. These components are implemented with various levels of decentralisation. Referencing certain aspects of design decisions defines the variations among these patterns and facilitates the acquisition of suitable designs. In the following section, we classify MAPE components based on their degree of decentralisation. Then, we further describe the various interaction patterns in this class of decentralisation.

2.2.2.1 Decentralised Control Approaches

MAPE feedback loops can be classified based on the interaction style between the loop elements and their peers. Decentralised approaches vary based on the level of collaboration among MAPE elements and their distributed peers. However, the decentralisation of some aspect of the control is a common factor among these various interaction styles. The decentralised approach is considered adequate when knowledge about the adapted system is inherently distributed. Precisely, the scale of partitioned knowledge leads to inefficiency in realising the adaptation requirements e.g. (cost, performance, and reliability). Another reason for applying decentralised approaches is the lack of an accountable trusted entity for collecting system data and acting accordingly. Consider for instance, a number competing agents in a multi agent system. Each agent has to work individually to increase the efficiency and reduce cost. These agents could also be competing companies where no trustworthy entity can fully adapt the system independently. In this scenario, each agent coordinates with one another to adhere to service level agreements and governing legislations.

As mentioned earlier, there are issues that necessitate the use of decentralised control approaches. However, there are two main sub-classes to describe the required level of coordination between the four MAPE activities with their corresponding peers. A definition of each class

accompanied by illustrative diagrams adapted from work by Weyns et al. [184], is presented below.

1. **Full collaboration**

   The first subclass of coordination demands a complete collaboration between each of the MAPE loop activities that manage the partitions of the managed system with corresponding peers of other MAPE loops. That is, due to the highly interrelated nature of the managed distributed components that require ongoing coordination between the managing components. Each managing component implements a set of MAPE activities, where each activity is linked to its peer activity in the other managers. In turn, the MAPE activities in each managing component collaborate with their corresponding peers to avoid an adverse adaptive behaviour. This undesired behaviour may occur when a local adaptive behaviour triggers an adaptive reaction in another managed component. A sequence of counteractive adaptive reactions may occur in the worst scenario, affecting the system functional and non-functional requirements. As depicted in Figure 2.8, each MAPE loop activity has to collaborate with its respective peers to eliminate the risk of faulty adaptive decision. This collaboration prevents sub-optimal adaptation decision from the overall system viewpoint. It also provides a consistent view of the state in all managed components. For instance, the analyse activity in one loop shares the findings with relevant peers to asses the need for adaptation, while the execute activity synchronises adaptive actions to avoid adverse effects on overall system requirements. As the example indicates, the collaboration mechanism can range from a simple exchange of information about the managed artefact to sophisticated coordination of actions.

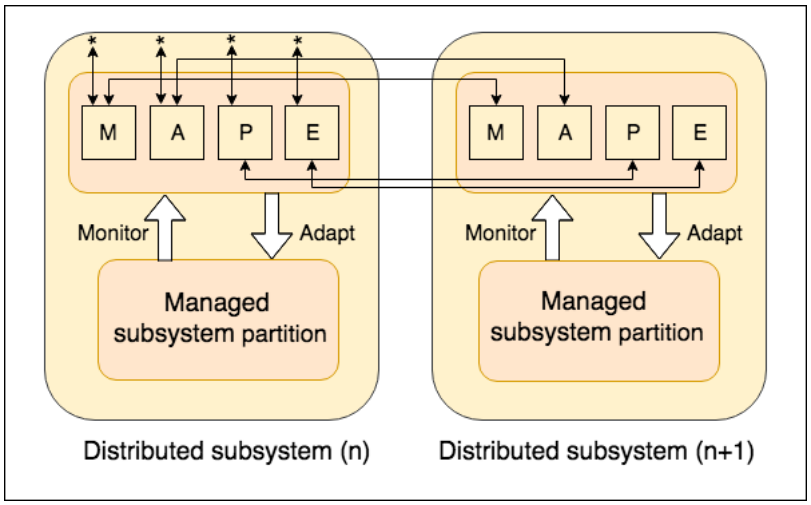

FIGURE 2.8. Coordinated control pattern adapted from [184].

Decentralised control approaches, in general, distribute the computation and communica-

tion load across managing components to expand the size of managed components. The increase of system size requires the reduction of computation and communication burden by localising and decreasing the interaction of MAPE control components with peers. The degree of correlation between MAPE control components could affect the expansion of system size. If MAPE activities interact with too many corresponding nodes, both the effort and cost to achieve adaptation will be high. This may cause delays in realizing adaptive decisions or overheads in the communication channel. On the other hand, less coordination may lead to inconsistent adaptation or even sub-optimal decisions.

2. **Partial coordination**

   The second subclass requires partial coordination, mainly because the managed components are loosely coupled. The decision raised from the adapting distributed system components does not influence changes in peer components. The coordination pattern in this class is limited to one activity (monitor) in the MAPE loop. In this scenario the managing components share the state of their managed resources through collaboration in the monitoring activity to adapt their managed resources. This approach restricts the collaboration to the monitor activity of the MAPE loop cycle. The state information about a particular component is exchanged to support the functional and non-functional adaptation in other system components. The information sharing can be explicit through exchange of messages, or implicit by means of observing the state of a managed resource. Aside from information exchange, decentralised MAPE loops operate in isolation in the rest of the loop activities (Analysis, Plan, Execute). Figure 2.9 illustrates the concept of partial coordination. For example, consider a smoke alarm device to detect fire hazards. A change in device state (e.g. smoke or fire detection) can trigger preventive adaptive actions in nearby appliances that share the same physical environment. In this case, each monitor activity in surrounding appliances observes the state of the smoke detector either by observing the state of the environment directly or by receiving the state of the environment by coordinating with another monitor activity that observes the environment's state.

   Since the interaction is limited to monitoring activities, this approach serves better from a scalability perspective than the previous approach that requires further coordination in remaining activities (Analyse, Plan, Execute). However, the full collaboration approach reduces the burden on communication and computation similarly. However, partial collaboration merely provides better scalability in terms of communication. Similarly, both approaches have to minimize MAPE interactions with corresponding peers to preserve scalability. The effect of intensive interaction is less severe in approaches that restrict coordination to the monitor activity. This restriction eliminates any inefficiencies in adaptation due to communication overheads in the remaining activities (Analyse, Plan, Execute). The main drawback of this approach is in the situation where a local adaptation decision may conflict with other control components and lead to a series of accidental adaptations.

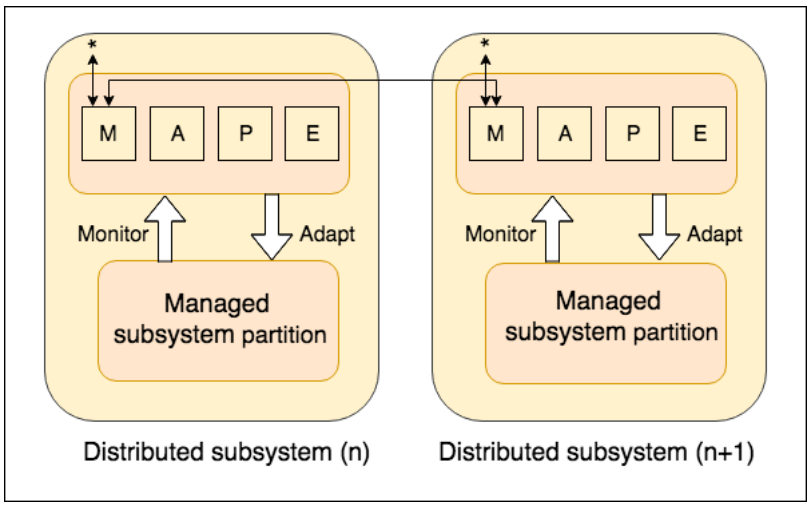

FIGURE 2.9. Partial coordinated control approach inspired by [184].

Likewise, limiting collaborations to the monitor phase can help to prioritize local adaptation requirements over global ones [184].

2.2.2.2 Hybrid Decentralised Control Approaches

In the following section, a brief discussion about the most frequently used approaches that combine both centralised/decentralised control approaches to address specific concerns will be provided.

1. **Master/Slave Approach -** Decentralised control in general promotes local adaptation by including a MAPE loop cycle in each of the distributed subsystems and collaboration with peers when possible to reflect on the system‘s global state on the local adaptation decisions. However, when the overall system adaptation requirements are regarded as crucial, the centralised pattern provides a consistent view of the system‘s overall state and requirements. This approach prioritises system-wide adaptation requirements by centralising the analyse and plan activities (Master) and decentralising the monitor and execute activities (Slave). As Figure 2.10 depicts, this architecture comprises a high-level adaptation logic that performs analysis and planing for two or multiple components that implement the monitor and execute activities. Each subordinate component supplies adaptations with relevant data to the centralised analysis activity in order to inspect aggregated data for a particular adaptation concern. Then, in case there is a need for adaptation, an adaptation plan is constructed and passed down to the relevant execute activity.

   The master/slave control approach succeeds in reflecting the state of the distributed components and prioritising overall system adaptation requirements. However, in the course of achieving this objective, centralising the analyse and plan activities may cause a communi-

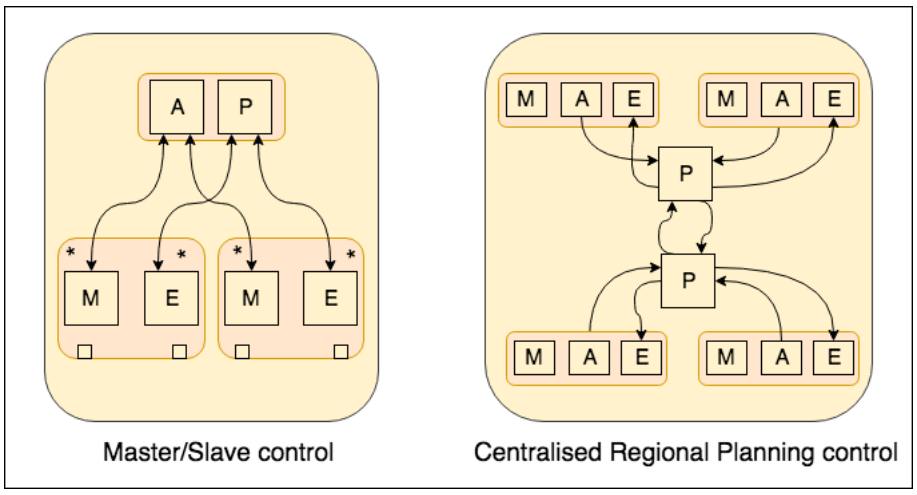

FIGURE 2.10. Hybrid Control Approach adapted from [184].

cation bottleneck. Especially, in the class of systems where the size of distributed managed components (Slaves) is large. Also centralising the portion of the distributed system leads to a single point of failure if the master adaptation logic fails to orchestrate the adaptation decision to subordinate components.

2. **Decentralised Regional Planning Approach**

   The decentralised regional planning approach is feasible for loosely coupled component systems where components are divided into segments (regions) that centralise the planning activity. Centralising the plan activity facilitates the realisation of adaptation within regions (locally) or across regional boundaries. This approach can be illustrated through an interconnected cloud infrastructure where utilising each segment of servers is associated with a cost. The goal of local adaptations (within regions) is to balance the resource utilisation. In addition, the objective of cross-boundary adaptation is to delegate excessive load to another region. The need of load delegation is determined locally within the regional planner and is acted upon in cross-boundary style. Unifying the plan activity regionally allows to adopt a different strategy for local regional adaptation and for the global ones. The reason for obtaining a different strategy in the cloud infrastructure example is the cost factor which segregates the servers that share the same cost between two regions. Figure 2.10 illustrates a centralised planning activity (MA**P**E) that plans adaptation actions for each region. The regional planner collects adaptation requests from a group of locally analysing activities to decide whether to act on these requests either locally or globally. Acting on the global adaptation plan requires interaction with the corresponding planner to coordinate adaptation actions that overrun their regional ruling. The moment a plan is finalised and agreed upon, the associated execute activities are invoked to adapt the necessary group of components.

According to Weyns et al. [184] this control approach permits a layered separation of concerns within each planning region. That is, all MAPE control loops that reside in the same region pass the planning activity to a centralised component. Each region is represented with a planning component that is accountable for adaptation within its boundary and coordination with respective planners when needed. Since the regional planner can act on adaptation on the local resource and global ones, Weyns et al. suggest that this control approach permits an additional horizontal separation of concerns. This approach also avoids centralising the analyse activity as in the Master/Slave control approach. Instead, it relies on local activities to examine monitored data and for reporting to the regional planner if adaptation is required. By localising the analysis activity, the interaction with the centralised plan activity becomes less frequent. However, in a large subsystem directed by a single plan activity, there is an overhead risk in computation and communication, by aggregating local analysis data and collaborating with peer planners. Also another risk may arise as a result of eliminating the coordination between execute activities. That is, the regional plan activity establishes a detailed plan to orchestrate the execution of adaptation actions.

2.2.2.3 Hierarchical Control Approach

MAPE control loops in distributed systems can be organised hierarchically for managing adaptation complexity. In this approach, MAPE loops are placed on different layers for establishing a layered separation of concerns. Each layer is themed with a specific concern at a certain level of abstraction. As in Kramer and Magee's architecture model [113], the top layer of control loops are concerned with goal management. They synthesise the global adaptation strategy based user requirements. Also, control loops at this level operate less frequently compared to the lower-layer loops. Control loops at the lowest layer often operate more frequently because they are in direct contact with the managed resources, while upper layers operate at longer time intervals. The middle-layer control loops manage and synchronise adaptation in the subordinate loops. The hierarchical control approach is feasible in highly-coupled distributed systems where adaptation requires constant interaction and collaboration to avoid adaptation conflicts. Figure 2.11 shows a possible instance of this control approach. There are three layers of control loops and it is possible to add more layers to increase the level of abstraction.

Using the hierarchical control approach can potentially resolve the complexity of the adaptation decision in distributed systems. The hierarchical top-down structure permits a layered separation of concerns where the bottom-layer control loops concentrate on specific adaptation concerns. This approach also allows the control loops in the upper layers to focus on wider perspectives and coordinate adaptation in the dependent loops. However, Weyns et al. state that, hierarchically organising the adaptation concerns is a non-trivial task. This is particularly difficult when adaptation requirements conflict with one another. Their work also referred to the

possibility of a certain scenario where the management of systems that require many control layers can become very complicated. They came to this conclusion after exploring similar hierarchical patterns in the domain of behaviour-based architectures [4]. In addition, de Lemos et al. [64] argue that hierarchical centralised approaches which are found in large-scale systems respond less effectively to adaptation concerns. That is explained by the fact that centralised decision mechanisms require exchanging all relevant data to adapt accordingly. de Lemos et al. also indicate that centralising the knowledge required for adaptive decision making may affect the scalability of the underlying managed system. In contrast to the hierarchical approach, decentralised control scales better in terms of communication and computation, because decentralised control loops rely on local information to adapt. This reliance on local information results in less intense interaction and distributed computation effort across system control loops. Also, hierarchical control approaches are considered less robust since a failure in hierarchical control loop could disrupt the adaptation feature at the subject system [34, 49, 64, 126].

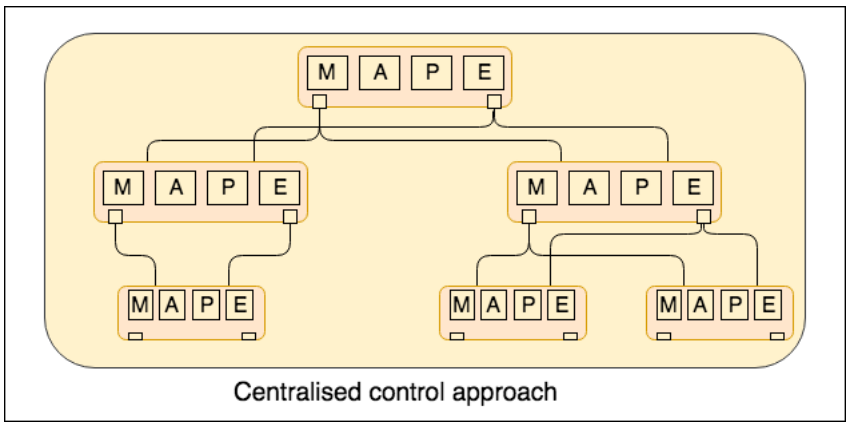

FIGURE 2.11. Centralised Control Approach.

2.3 Markovian Models

Many approaches in self-adaptation employ mathematical techniques for analysing the behaviour of self-adaptive systems with strict requirements. Markovian models are relevant in this context because it provide the mathematical framework for analysing the behaviour of such systems. In this section, we are going to introduce several variants of Markov models. A Markov model is a transition system in which the move between states is determined by a probabilistic choice. A Markov model can be employed to model systems that exhibit stochastic behaviour. The stochastic behaviour of such systems is modeled as a stochastic processes in which the transition among theses processes is specified as a family of random variables defined as:

$$\{X(t),\ t \in T\}$$

The behaviour of the stochastic process is defined as a collection of random variables $X(t)$ defined in a probability space and indexed by time such that the collection yields the the state (i.e, the value of the random variable) of the process at time $t \in T$. T is a set of time points that are discrete ticks of time $T = \{0, 1, 2, \ldots\}$ in a *discrete-time* stochastic process, or continuous time $T = \{t : 0 \leq t < \infty\}$ in a *continuous-time* process. Thus, $X(t)$ denotes the current state at time t and the term *state space* defines the set of all possible states given by $X(t)$. If the state space S is considered discrete such that the states in a stochastic process are defined as a set of discrete points $S \in \mathbb{N}$, the stochastic process is described as a chain. Moreover, if the stochastic processes satisfy the Markov property (Definition 2.1) then the processes are referred to as Markov processes [153]. The stochastic processes are said to satisfy the Markov property when the future evolution of the transition system depends solely on the current state rather on the history of previously encountered states.

Definition 2.1. A stochastic process $\{X(t) \mid t = 0, 1, \ldots\}$ comprises of a series of successive states $s_0, \ldots, s_t$ which satisfy the Markov memoryless property when

$$P[X(t) = s_t \mid X(t-1) = s_{t-1}, \ldots, X(0) = s_0] = P[X(t) = s_t \mid X(t-1) = s_{t-1}]$$

2.3.1 Discrete-Time Markov Chains

A discrete-time Markov chain is a state-transition system where transitions between states are specified by a probabilistic distribution and the timing of these transitions occur in discrete steps separated by a regular time interval. The state-transition system can only hold one state at any point of time and the transitions from source to target states are regulated by a discrete probabilistic distribution.

2.3.1.1 Definition

A definition of a Discrete Time Markov Chain (DTMC) adapted from [118] is illustrated as follows.

Definition 2.2. A labelled Discrete Time Markov Chain (DTMC) D is a tuple (S, s_i, P, AP, L) where

1. S is a non-empty, finite set of states;
2. s_i is an initial state and $s_i \in S$;
3. $P: S \times S \to [0, 1]$ is a transition probability matrix such that the sum of outgoing transitions for all states:

$$\sum_{s' \in S} P(s, s') = 1, \ \forall \ s \in S; \tag{2.1}$$

4. AP is a set of atomic propositions that are valid in the state $s \in S$ and L: $S \rightarrow 2^{AP}$ is the labelling function that assigns the atomic propositions AP to each state.

The atomic propositions facilitate reasoning about relevant characteristics of a state-transition system. Atomic propositions along with labelling of states provide a means to declare the required characteristic in simple statements that are associated with system states and can be evaluated to true or false with the current state of the system. Consider the system shown in Figure 2.12 where we can use atomic propositions to encode simpler statements. For instance, " what is the probability of reaching in the future *success* state".

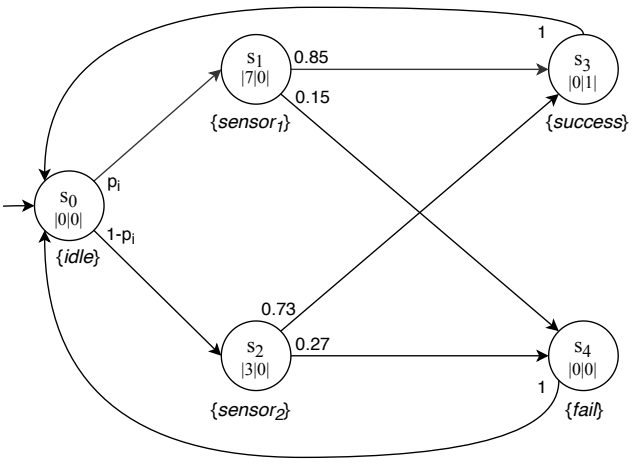

Figure 2.12: An example of a DTMC model augmented with state rewards and costs

Example 2.1. Consider an Internet of Things (IoT) system comprising two sensors that can take periodic measurements of the environment. Figure 2.12 illustrates a DTMC model that represents the behaviour of the $n = 2$ sensors. The system can be configured using a probability parameter (p_i) to utilise either of the sensors $sensor_1$ and $sensor_2$. The first sensor ($sensor_1$) operates with higher probability (0.85) to suggest that measurements taken are sufficiently accurate. While the second $sensor_2$ is less effective in producing accurate readings as its probability is at (0.73). A successful accurate reading results in reaching state s_3 (*success*), while a failure in providing an adequate reading leads to state s_4 (*fail*). Thus, the DTMC model shown in Figure 2.12 is

described as follows:

$$
\begin{aligned}
S &= \{s_0,\ s_1,\ s_2,\ s_3,\ s_4\} \\
AP &= \{idle,\ sensor_1,\ sensor_2,\ success,\ fail\} \\
s_i &= \{s_0\} \qquad\qquad L(s_0) = \{idle\} \qquad L(s_1) = \{sensor_1\} \\
L(s_2) &= \{sensor_2\} \qquad\qquad L(s_3) = \{success\} \qquad L(s_4) = \{fail\}
\end{aligned}
$$

$$
P = \begin{bmatrix}
0 & p_i & (1-p_i) & 0 & 0 \\
0 & 0 & 0 & 0.85 & 0.15 \\
0 & 0 & 0 & 0.73 & 0.27 \\
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

2.3.1.2 Extending DTMCs with rewards

The states and transitions of a DTMC model can be augmented with positive real-valued quantities that can be interpreted as either rewards or costs. These rewards and cost values support the quantitative analysis of relevant system characteristics which aim to maximise or minimise. The values also support analysis of properties for meeting a certain threshold.

Definition 2.3. Consider a DTMC model D is a tuple (S, s_i, P,AP, L), a reward structure is a pair of real-valued functions ($\underline{\rho}, \iota$), where

1. $\underline{\rho} : S \to \mathbb{R}_{\geq 0}$ is a state reward function $\underline{\rho}(s)$ that defines the real-value reward/cost associated being in state $s \in S$ for one discrete time interval.

2. $\iota : S \times S \to \mathbb{R}_{\geq 0}$ is a transition reward function that defines the reward/cost received on each transition in the model.

Example 2.2. The software system described in Example 2.1 could be augmented with cost and reward values to support the quantitative analysis of the relevant system characteristics. Thus, in this example we assume that the cost (such as, energy consumed or time units) associated with using $sensor_1$ is higher than using $sensor_2$. This is evident in the cost augmented with invoking the state s_1 ($sensor_1$) which cost 7 energy units as annotated below the state definition in Figure 2.12, while calling the second state s_2 $sensor_2$ costs 3 units. Depending on the application objectives, the goal could maximise accurate reading rewards by reaching state s_3 (*success*) more frequently and meeting an energy budget constraint (e,g. cost associated with visiting states $sensor_1$ and $sensor_2$) at the same time. In this instance, the cost and rewards are associated with the states rather than with transitions. These costs and rewards are labeled below the state definition and take the following form:

$$| < cost\ of\ invoking\ sensor > | < reward\ for\ an\ accurate\ reading > |$$

In general, the difference between cost and reward is semantic, as the mathematical quantitative evaluation of both quantities is similar, but the semantic interpretation of each differs. The DTMC model described earlier in Example 2.2 is augmented to describe cost and reward associated with visiting states as follows:

$$\underline{\rho}^{reward} = (0,0,0,1,0) \qquad \underline{\rho}^{cost} = (0,7,3,0,0)$$

As illustrated above, the cost ($\underline{\rho}^{cost}$) and reward ($\underline{\rho}^{reward}$) quantities are assigned to states rather than to the transition between the states. In this example there are no costs and rewards associated with transitions between states, and therefore the definition for ι is $\iota = 0_{5,5}$, where $0_{5,5}$ is a 5×5 matrix and the value of its elements is 0.

2.3.2 Continuous-Time Markov Chains

A **Continuous-Time Markov Chain (CTMC)** is a stochastic state transition model that comprises of discrete states where transitions among states occur in continuous time (e.g., any point of time) rather than at discrete time intervals, as in a DTMC model. Similar to DTMC, CTMC assumes the memory-less property of a Markov model in which the transition to next state depends solely on the current state and not on the history of previously visited states. Since transitions occur in continuous time and can be defined using exponential probability distributions [10, 118], the CTMC approach is feasible for modelling QoS characteristics of a system. Characteristics such as reliability and performance of real-time systems can be modeled and thus the behaviour that corresponds to these characteristics can be analysed. The analysis of a real-time system behaviour can be carried out to reason about the instant behaviour at a particular time instance or the long term behavior.

2.3.2.1 Definition

Definition 2.4. A labelled Continuous Time Markov Chain (CTMC) C is a tuple (S, s_0, R,AP, L) where

1. S is a nonempty set of finite states;
2. s_0 is an initial state and $s_0 \in S$;
3. R: $S \times S \rightarrow \mathbb{R}_{\geq}0$ is the transition rate matrix;
4. AP is a set of atomic propositions that are valid in the state $s \in S$ and L: $S \rightarrow 2^{AP}$ is the labelling function that assigns the atomic propositions AP to each state.

Transitions in a CTMC are controlled by the transition rate matrix R, which defines the probability that a transition between a given state s, $s' \in S$ may occur within t time units if

$R(s,\ s') > 0$. The transition probability of $(s,\ s')$ within t time units is given by $(1 - e^{-tR(s,\ s')})$. The rate at which a transition may be triggered has an exponential distribution in which the time required by the state s to move to s' is independent of the time taken to wait for the transition to occur. The rate of transition defines the time t required by state s before a transition can occur with an *exit rate*

$$E(s) = \sum_{s' \in S} R(s,\ s') \tag{2.2}$$

The previous formula defines the rate at which a transition may occur. However, the destination of the transition is an outcome of a probabilistic choice. The probabilistic transition choice is independent of the time already spent in state s and provided by the embedded DTMC of the CTMC. Once a state s has been left, the probability of being in state s' is defined by the embedded DTMC as follows:

Definition 2.5. The embedded DTMC D of a CTMC $C = (S,\ s_0,\ R, AP,\ L)$ is given by the DTMC

$$emb(C) = (S,\ P_{emb(C)},\ s_0,\ AP,\ L))$$

where for a given $s,\ s' \prime S$

$$P^{emb(C)}(s,\ s') = \begin{cases} \frac{R(s,s')}{E(s)} & \text{if } E(s) \neq 0 \\ 1 & \text{if } E(s) = 0 \text{ and } s = s' \\ 0 & \text{otherwise} \end{cases}$$

Definition 2.5 illustrates the expected behaviour of the CTMC once a transition from state s is about to happen with an *exit rate* $E(s)$. This defines the time that is assumed to have an exponential distribution before leaving the state. The anticipated behaviour is an instantaneous move to state s' with a probability specified by $P^{emb(C)}(s,\ s')$.

2.3.2.2 Extending CTMCs with rewards

A CTMC can be augmented with quantitative costs and rewards similarly to a DTMC. However, the costs and rewards associated with staying in a given state are determined based on the time spent while visiting that state.

Definition 2.6. Consider a CTMC model C is a tuple $(S,\ s_0,\ R, AP,\ L)$, a reward structure of C is a pair of real-valued functions $(\underline{\rho}, \iota)$, where

1. $\underline{\rho} : S \to \mathbb{R}_{\geq 0}$ is a state reward function $\underline{\rho}(s)$ which specifies the rate at which the reward is received while the CTMC remains in state $s \in S$;

2. $\iota : S \times S \to \mathbb{R}_{\geq 0}$ is a transition reward function that defines the reward/cost received on each transition in the CTMC.

Example 2.3. Let us consider extending the system illustrated in Example 2.1 to continuous time transition rather than discrete steps of transition as in a DTMC. The transition system shown in Figure 2.13 describes the mean time in seconds spent in system states. This is evident in the transition rate which labels each edge of transition between states. The system attempts to take a periodic sensor measurement by transitioning from state *idle* with a rate of 60. The previous rate defines the mean time spent in state s_0 before the system attempts to take a measurement. Similarly sensors $sensor_1$ and $sensor_2$ operate with rates 30 and 40 respectively. $sensor_1$ provides more measurements per time ratio than $sensor_2$ i.e. higher throughput. However, this consumes more energy units as the reward associated with staying in state is 3.5, while the reward for being in state $sensor_2$ is 1.2. In contrast to how rewards are identified for visiting a state in a DTMC, rewards in a CTMC are identified as the product of multiplying the reward associated with being a state with time spent visiting that state. Since CTMC factors time in state reward calculation, the reward that corresponds with taking a successful sensor measurement is associated with transition edges to *success* state. Thus, the CTMC model shown in Figure 2.13 is described as follows:

$$
\begin{aligned}
S &= \{s_0,\ s_1,\ s_2,\ s_3,\ s_4\} \\
AP &= \{idle,\ sensor_1,\ sensor_2,\ success,\ fail\} \\
s_i &= \{s_0\} \qquad\qquad L(s_0) = \{idle\} \qquad L(s_1) = \{sensor_1\} \\
L(s_2) &= \{sensor_2\} \qquad\qquad L(s_3) = \{success\} \qquad L(s_4) = \{fail\}
\end{aligned}
$$

$$
R = \begin{bmatrix}
0 & 30 & 30 & 0 & 0 \\
0 & 0 & 0 & 21.9 & 8.1 \\
0 & 0 & 0 & 34 & 6 \\
100 & 0 & 0 & 0 & 0 \\
100 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

2.3.3 Markov Decision Processes

Unlike a DTMC, the Markov decision process (MDP) [120] provides a means to model the probabilistic behaviour of a software system that incorporates non-deterministic behaviour. Modeling the non-deterministic behaviour aids in defining certain aspects of a system's behaviour which are considered unknown and cannot be represented as a probabilistic choice. Example of such behaviour in modeling a system comprises the ordering of action executions of two parallel components that execute tasks in a *concurrent* manner. Another example is when a controller needs to devise a strategy consisting of a series of actions synthesised at runtime for solving a problem that is modeled by non-deterministic choices.

The later example illustrates the use of an MDP model to solve the strategy synthesis problem in which the goal for a given MDP model M is to locate some *strategy* that satisfies a specified property. Strategies describe various possible ways of resolving the non-deterministic choices to

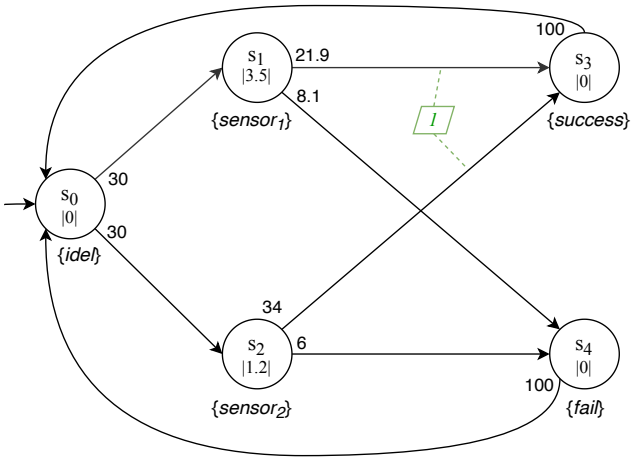

Figure 2.13: An example of a CTMC model augmented with state rewards and costs

meet a specified property. These non-deterministc choices are considered as *under specifications* from a design point of view and left for the controller to synthesise strategies based on the range of scenarios encountered at runtime. Also, from the various possible strategies, we can obtain an optimal strategy which optimises a specified objective.

In the context of this thesis, we focus on the use of an MDP to address the strategy synthesis problem. Solving this problem is carried out by employing probabilistic model checking which provides the ability to analyse quantitative properties in an exhaustive manner. The use of MDP to handle strategy synthesis has received growing interest in areas such as synthesis controllers for robotic applications [135, 188], planning under uncertainty [42, 111], and synthesis strategies for penetration attack of a software system [164]. The definition of a Markov Decision Process (MDP) presented hereafter is adapted from [120].

2.3.3.1 Definition

Definition 2.7. A Markov decision process (MDP) is a tuple $M = (S, s_0, A, \delta_M, L)$ where

1. S is a nonempty set of finite states;

2. s_0 is an initial state and $s_0 \in S$;

3. A A is a finite set of actions;

4. δ_M: $S \times A \rightarrow$ Dist(S) is is a (partial) probabilistic transition function, mapping state-action pairs to probability distributions over S;

5. AP is a set of atomic propositions that are valid in the state $s \in S$ and L: $S \rightarrow 2^{AP}$ is the labelling function that assigns the atomic propositions AP to each state.

Beginning from an initial state s_0, an MDP model describes the possible course through which the state of a system may evolve. For a given state $s \in S$, $A(s) \subseteq A$ identifies the possible set of enabled actions from state s. For each action $a \in A$, there is an embedded probability distribution defined as $A(s) \stackrel{\text{def}}{=} \{a \in A \mid \delta_M(s, a)\}$ such that when action s is selected, the transition to the next state s' is determined by the probability distribution $\delta_M(s,a)$. The choice of which action $a \in A$ to select in an MDP model is considered to be a non-deterministic choice.

For an MDP model M, a *path* (ω) comprises of a possible finite or infinite sequence of transitions $s_0 \stackrel{a_0}{\rightarrow} s_1 \stackrel{a_1}{\rightarrow}$, where for all $i \in \mathbb{N}$ states in the sequence $s_0, s_1, \ldots, s_i \in S$, there is an associated action $a_i \in A(s_i)$ such that $\delta_M(s_i, a_i)(s_{i+1}) > 0$. $\omega[i]$ denotes the i-th state of path ω and last(ω) indicates the final state if ω consists of a finite sequence of states. For a given MDP model M, $\text{FPath}_{M,s}$ indicates the set of all finite paths beginning in s. Similarly, $\text{IPath}_{M,s}$ denotes the set of all infinite paths starting in s.

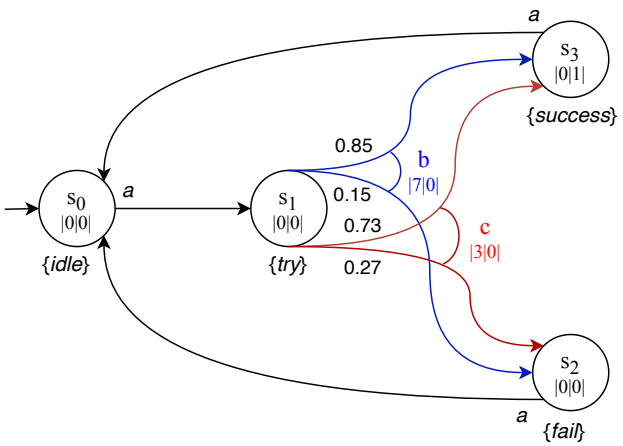

Figure 2.14: An example of a MDP model augmented with state rewards and costs

Example 2.4. Figure 2.14 presents a MDP M, which describes the behavior of a robot equipped with two sensors that can take periodic measurements of the robot's environment. The model comprises four states and for each of these states, there are one or more actions available from the set $A = \{a, b, c\}$ that allows the robot to use either of the available sensors. In state s_1, the robot chooses in a non-deterministic manner any of its two sensors, which are represented as action b and c respectively. When either of b and c actions is triggered, the robot utilises the corresponding sensor to provide a measurement. Once an action is triggered, a probabilistic transition occurs to state s_2 if the measurement fails to meet some accuracy criteria or otherwise to state s_3 if the taken measurement is of sufficient accuracy. Action a which is found in the rest of states entails a probabilistic transition to an alternative state. As shown in Figure 2.14, each transition arrow is labeled with a probability, while if the probability is 1, the label is omitted.

The MDP model shown in Figure 2.14 is described as follows:

$$
\begin{aligned}
S &= \{s_0, s_1, s_2, s_3\} \\
AP &= \{idle, try, success, fail\} \\
A &= \{a, b, c\} \\
s_0 &= \{s_0\} \qquad\qquad L(s_0) = \{idle\} \qquad L(s_1) = \{try\} \\
L(s_2) &= \{fail\} \qquad\qquad L(s_3) = \{success\}
\end{aligned}
$$

$$
\delta_M = \left[\begin{array}{cccc}
0 & 1 & 0 & 0 \\ \hline
0 & 0 & 0.15 & 0.85 \\
0 & 0 & 0.27 & 0.73 \\ \hline
1 & 0 & 0 & 0 \\ \hline
1 & 0 & 0 & 0
\end{array}\right]
$$

2.3.3.2 Extending MDPs with Rewards

An MDP can be augmented with *rewards* to support the modelling of various quantitative measures. The reward mechanism is used to include quantitative measures that should be maximised (e.g., utility). The same mechanism can be utilised to define costs which comprise of the quantitative measures that we need to minimise (e.g., time, energy consumption). Although, rewards can be attached to states as in a DTMC, however, in the case of MDP, rewards can also be appended to actions as shown in Figure 2.14.

Definition 2.8. For a Markov decision process (MDP) defined as a tuple $M = (S, s_0, A, \delta_M, L)$, a reward structure for M is defined as a function that take the form $r : S \times A \to \mathbb{R}_{\geq 0}$.

2.3.3.3 Strategies

A *strategy*, which is also called policy or adversary, depending on the context, is used to reason about the behavior of an MDP [120]. A strategy is considered as a recipe for resolving non-determinism in an MDP by means of defining which action or series of actions need to be chosen for each state. In general, the choice of action can be made randomly and it can also be made based on choices that have occurred in previous executions of the MDP.

Definition 2.9. For a Markov decision process (MDP) defined as a tuple $M = (S, s_0, A, \delta_M, L)$, A strategy of M is defined as a function that take the form $\sigma : \text{FPath}_M \to \text{Dist}(A)$ such that $\sigma(\omega)(a) > 0$ only if $a \in A(\text{last}\omega)$.

Strategies are classified in terms of the mechanism in which an action is chosen that can be randomised or depends on the memory record of the past MDP's execution so far. Such classification is useful as it reduces our attention to a certain class of strategies of $\sum_M$ which denotes a set of all strategies of M.

1. Randomised strategy: A strategy σ is considered to be deterministic if for all finite paths of M, $\sigma(\omega)$ is a point distribution for all $\omega \in \text{FPath}_M$. Otherwise the strategy is considered to be a randomised strategy.

2. Memory strategy: A strategy σ is said to be based on finite-memory if each action that occurs in a mode of strategy execution of $\sigma(\omega)$ is recorded and plays role along with last(ω) of the previous finite modes. Otherwise, σ is considered to be memory-less as it relies only on last(ω).

The behaviour of M is said to be fully probabilistic if a certain strategy σ is acquired such that all nondeterminism in M is resolved. In Section 2.4.1, we introduce the temporal logic used to specify properties of an MDP. Then, we explain how the specified properties of the MDP can be verified to check whether the synthesised strategy meets the specified properties.

2.4 Probabilistic Temporal Logics

This section introduces variants of probabilistic temporal logics which is a set of specification languages that allows to formally specify the QoS properties of interest in a software system whose behaviour can be modelled using one of the Markovian modelling formalisms from the previous section (i.e., DTMCs, CTMCs or MDPs). Having a formal specification of the QoS properties helps to reason about the behaviour of a software system over time. This section presents Probabilistic Computation Tree Logic (PCTL) [21] which is a temporal logic variant used to specify properties of DTMC and MDP. Also the section introduces Continuous Stochastic Logic (CSL) [9] which is another temporal logic variant used to define properties of CTMC.

2.4.1 Probabilistic Computation Tree Logic

Probabilistic computation tree logic (PCTL) [91] is a language that enables the formal specification of probabilistic properties of a DTMC and an MDP. Through model checking engines such as Prism [119], Storm [65], and MRMC [104], we can formally verify if the behaviour of a software system, which represented in the relevant stochastic model i.e., DTMC or MDP, satisfies the specified properties. PCTL logic is based on Computation Tree Logic (CTL) [52] and allows to evaluate if a specified property holds within a certain discrete steps of time.

Definition 2.10. Below is the grammar used in PCTL to express a state formula Φ and a path formula Ψ;

$$\begin{aligned} \Phi &::= true \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid P_{\triangleright\triangleleft p}[\Psi] \\ \Psi &::= X\ \Phi \mid \Phi\ \cup^{\leq K}\ \Phi \end{aligned} \tag{2.3}$$

and the cost/reward augmented PCTL state formulae are defined by the grammar;

$$\Phi ::= R_{\triangleright\triangleleft r}[C^{\leq K}] \mid R_{\triangleright\triangleleft r}[I^{=K}] \mid R_{\triangleright\triangleleft r}[F\ \Phi] \tag{2.4}$$

where Φ and Ψ denote the state and path formulae respectively, $a \in AP$ is atomic proposition, $\bowtie$ is a relational operator and $\bowtie \in \{<, \leq, \geq, >\}$, $p \in [0,1]$ is a probability bound, $k \in \mathbb{N}_{>0} \cup \{\infty\}$ and r is defined as a reward bound $r \in \mathbb{R}^{\geq 0}$.

A PCTL formula is defined with reference to states in a model i.e. DTMC or MDP. For instance, a formula $P_{\bowtie p}[\Phi]$ which comprises the probabilistic operator P can be utilised to assert if a state formula Φ holds if the probability of the future evolution of the system satisfies the bound $\bowtie p$. A path formula Ψ can only be specified within a probabilistic operator P. Defining an execution path Ψ in a probabilistic operator provides bounds on the probability of system evolution. The next formula $X\ \Phi$ can be used to assert if there is an execution path which has a next state that satisfies Φ. The *bounded until* formula $\Phi_1\ \cup^{\leq K}\ \Phi_2$ is used to check if an execution path continuously satisfies Φ_1 for all previous discrete interval steps (x) before Φ_2 becomes true at interval step $x \in K$.

Similarly, in the following we list examples of the use of the reward operator R in PCTL semantics.

- $R_{\bowtie r}[C^{\leq K}]$: The reward operator $R_{\bowtie r}$ used to assert if the total accumulated rewards along a path Ψ satisfies the defined bound $\bowtie r$ within k interval steps.
- $R_{\bowtie r}[I^{=K}]$: The reward operator verifies if the anticipated reward at step k meets the bound specified by $\bowtie r$.
- $R_{\bowtie r}[F\ \Phi]$ The amount of reward accumulated up to satisfying the state formula Φ meets the bound specified by $\bowtie r$.

Analysing properties of a DTMC A DTMC model provides a formal description of the behaviour of the underlying system. The mathematical evaluation of system behaviour necessitates the incorporation of a precise language to support the formal specification of probabilistic properties. The formal definition of system properties is utilised to verify through model checking tools and techniques if a system model D fulfills a specification ϕ such that;

$$D \models \phi \tag{2.5}$$

Since the transition among states are governed by a probability distribution, this leads to various combinations of possible execution paths starting from a given initial state. These possible execution paths along with concrete states are used to define a property and consecutively assert if it holds. For a given model D, a set of possible execution paths for a pair of states (s, s') is denoted as $Path^D(s)$. The behaviour analysis of the model D requires finding the unique probability $\mathbf{P}r_s$ on all paths $Path^D(s)$. The probability $\mathbf{P}r_s$ is the outcome of the sum of all the probabilities of all the paths $Path^D(s)$ that lead to state s' from the state s [118]. An execution path Ψ is defined as an non-empty finite or infinite set of consecutive states $\{s_0, s_1, \ldots\}$, where $s_i \in S$. The probability of transition $P(s_i, s_{i+1})$ in the sequence of states needs to be > 0 for all

$i \geq 0$. $\Psi(i)$ denotes the i-th state in the execution path and $|\Psi|$ denotes the length of the finite sequence of states in the path.

A satisfaction relational operator ($\models$) is used to define the semantics of PCTL. The semantics are expressed in reference to states $s \in S$ and paths $\Psi \in Path^D(s)$ provided in a DTMC model. For instance, consider the following notations:

- $s \models \neg\Phi$: indicates that the state formula $\neg\Phi$ is satisfied in state s.
- $s \models a$: means that the atomic proposition a is satisfied in state s if $a \in L(s)$
- $s \models \Phi_1 \wedge \Phi_2$: states that the state formulas Φ_1 and Φ_2 are both satisfied in state s.

Example 2.5. In the following, we extend the software system presented in Example 2.2 to support verifying the system behaviour against requirements defined using PCTL notations.

- $P_{\geq 0.95}[F^{\leq 300} success]$: This property checks if the probability that the system reaches *success* state within 300 steps exceeds the defined bound ≥ 0.95.
- $P_{\leq 0.95}[\neg success\ U\ fail]$: The probability that the system would produce more frequently inaccurate sensor readings before reaching for the first time a success state is $\leq 95\%$.
- $R^{reward}_{\geq 15}[C^{\leq k}]$: The expected amount of accurate reading accumulated up to reaching time step k is more than 15.

Analysing properties of a MDP For a given MDP M, the problem is to check if a strategy σ of M satisfies the property formula ϕ when starting from state s. We can use the satisfaction relation $\models$ to express the problem as $M, s, \sigma \models \phi$. As we are often interested in the behaviour of M starting form the initial state s_0 of M, the problem defined earlier is expressed as $M, \sigma \models \Phi$ instead of $M, s_0, \sigma \models \phi$. The previous problem verifies if M satisfies ϕ under a certain strategy σ. However, if we wish to verify if M satisfies ϕ for all possible strategies of M, then the problem is formally defined as per the below definitions.

Definition 2.11. For an MDP M and property ϕ, the problem is to verify for all possible strategies $\sigma \in \sum_M$? that $M, \sigma \models \phi$ holds.

This verification problem is often referred to in practice as the dual problem of *strategy synthesis* [120] in which the problem is twofold. The first is concerned with identifying some strategies of M that meet a property ϕ. The second part is concerned with finding the optimal strategy that optimises a specified objective.

Definition 2.12. For an MDP M and property ϕ, the strategy synthesis problem is to identify if exists, a strategy $\sigma \in \sum_M$ such that $M, \sigma \models \phi$.

In essence, the strategy synthesis problem is a verification problem in which we need to verify a property ϕ on MDP M, but the synthesis problem entails identifying *optimal values* using either of the following probabilistic or expected reward operators.

$$Pr^{\min}_{M,s}(\psi) = \sigma \in \sum_{M}^{\inf}\{Pr^{\sigma}_{M,s}(\psi)\} \qquad \mathbb{E}^{\min}_{M,s}(\text{rew}(r,\rho)) = \sigma \in \sum_{M}^{\inf}\{\mathbb{E}^{\sigma}_{M,s}(\text{rew}(r,\rho))\}$$

$$Pr^{\max}_{M,s}(\psi) = \sigma \in \sum_{M}^{\sup}\{Pr^{\sigma}_{M,s}(\psi)\} \qquad \mathbb{E}^{\max}_{M,s}(\text{rew}(r,\rho)) = \sigma \in \sum_{M}^{\sup}\{\mathbb{E}^{\sigma}_{M,s}(\text{rew}(r,\rho))\}$$

Where ψ and ρ define the objective that needs to be maximised or minimised and the value of the objective that corresponds to each operator, i.e., the probabilistic $Pr[\psi]$ or expected reward $\mathbb{E}[\rho]$ objectives, are derived from the grammar presented in Definition 2.10. For instance, consider a dual problem that comprises of a property $\phi = P_{\leq p}[\psi]$ that needs to be verified against M and at the same time we wish to synthesise a strategy for $\phi' = P_{\geq p}[\psi]$. This problem can be solved by evaluating $Pr^{\max}_{M}$, in which ϕ is satisfied by M when $Pr^{\max}_{M}(\psi) \leq p$ holds, and we are able to identify a strategy σ that satisfies ϕ' by computing $Pr^{max}_{M}(\psi) \geq p$. As we are often interested in finding an optimal strategy i.e., one that yields the maximum value in this example, the fixed bound p of the probabilistic operator is replaced by an alternative syntax $P_{\max=?}[\psi]$ which supports *numerical queries*.

Definition 2.13. Let ψ and ρ be the probability and reward objectives which can be derived using the grammar described in Definition 2.10. An optimal value for the probability/reward objective can be computed by using a numerical query that takes the form:

$$P_{\min=?}[\ \psi\ ] \qquad P_{\max=?}[\ \psi\ ] R_{\min=?} \qquad [\ \rho\ ] R_{\max=?} \qquad [\ \rho\ ]$$

Example 2.6. We now extend the robotic software system presented in Example 2.4 to synthesise a strategy satisfying a reach-ability property $P_{\leq 0.15}[F\text{fail}]$. The aim is to identify if there exists a strategy and can be achieved by evaluating the a numerical query $P_{\min=?}[\ F\ fail\ ]$. The computation yields the probability value 0.15 which corresponds to the memoryless deterministic strategy that chooses action (a) from state *try* which utilises $sensor_1$ as it carries out measurement with the least probability of failure (0.15). Another example to consider is in identifying an optimal strategy for minimising energy cost before reaching a *success* state. To find the optimal strategy, we use an expected reward property $R^{\text{cost}}_{\min=?}[\ F\ success\ ]$ that takes the form of a numerical query. The property uses a reward structure called *cost*, which as shown in Figure 2.14, maps the energy cost of using either of the two actions b and c, i.e., each of the two actions is mapped to a robot sensor. The resulting optimal strategy chooses action c i.e., the second sensor of the robot, from state *try*, and has the optimal value 4.12. This optimal strategy is once more memoryless and deterministic similar to the optimal strategy from the previous probabilistic property example.

2.4.2 Continuous Stochastic Logic

Continuous Stochastic Logic (CSL) allows specifying properties for CTMC models. Similar to non-probabilistic Computation Tree Logic, CSL is a branching time logic but it provides support for a probabilistic operator P. CSL, introduced by Aziz [8], is an extension of Probabilistic Computation Tree Logic (PCTL) [91], which was introduced by Hannson and Jonsson. Baier et al. added to CSL the until operator [12] which facilitates defining a time bound for evaluating CTMC properties.

Definition 2.14. Below we illustrate the grammar that dictates the syntax of Continuous Stochastic Logic (CSL):

$$\begin{aligned} \Phi &::= true \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid P_{\rhd\lhd p}[\Psi] \\ \Psi &::= X\ \Phi \mid \Phi\ \cup^{\leq I}\ \Phi \end{aligned} \tag{2.6}$$

and the cost/reward augmented CSL state formulae are defined by the grammar;

$$\Phi ::= R_{\rhd\lhd r}[C^{\leq T}] \mid R_{\rhd\lhd r}[I^{=T}] \mid R_{\rhd\lhd r}[F\ \Phi] \tag{2.7}$$

where Φ and Ψ denote state and path formulae respectively, $a \in AP$ is atomic proposition, $\rhd\lhd$ is a relational operator and $\rhd\lhd \in \{<, \leq, \geq, >\}$, $p \in [0,1]$ is a probability bound, $k \in \mathbb{N}_{>0} \cup \{\infty\}$ and r is defined as a reward bound $r \in \mathbb{R}^{\geq 0}$. Nevertheless, $I \in \mathbb{R} \geq 0$ and $T \in \mathbb{R} \geq 0$ used to find the expected reward at an interval of time and at a time instant respectively.

A CSL formula is defined in reference to the states of a CTMC model. Most of the CSL semantics that include path formulae are interpreted in a similar manner to PCTL. The only exception is for interpreting the *until* operator U when it handles the interval parameter $I \in \mathbb{N}_{>0}$. Similar to PCTL, a formula that includes the probabilistic operator P and has follow the semantic $P_{\rhd\lhd p}[\Phi]$, is used to assert if a state formula Φ holds if the probability of the future evolution of the system satisfy the bound $\rhd\lhd p$. A path formula Ψ can only be specified within a probabilistic operator P. Defining an execution path Ψ in a probabilistic operator provides bounds on the probability of system evolution. The next formula $X\ \Phi$ is used to assert if there is for a given execution path Ψ, a next state that satisfies Φ. A formula that includes the *time-bounded until* operator and follows the semantic $\Phi_1\ \cup^{\leq I}\ \Phi_2$ holds. If at any time instance within the interval $I = [0, \mathbb{R}_{>0}]$, there is a path which continuously satisfies is used to check if an execution path continuously satisfies Φ_1 before Φ_2 becomes true. However if the time interval is unbounded $I = [0, \infty]$ then the *time-bounded until* formula becomes *unbounded until* one. For formula that includes the probabilistic operator and take the form $P_{\rhd\lhd p}[\Phi]$, the probability bound defined by $\rhd\lhd p$ can be substituted with =? to find the probability bound which satisfies the path formula Φ.

In the following, we describe the different forms to express properties that include the reward operator R:

- $R_{\rhd\lhd r}[C^{\leq T}]$: The expression is used to assert if the total accumulated rewards satisfy the defined bound $\rhd\lhd r$ up to time T.

- $R_{\rhd\lhd r}[I^{=T}]$: The reward operator verifies if the anticipated reward at time instant T meets the bound specified by $\rhd\lhd r$.
- $R_{\rhd\lhd r}[F\ \Phi]$ The amount of reward accumulated up to satisfying the state formula Φ meet the bound specified by $\rhd\lhd r$.

Similar to PCTL, the reward bound $\rhd\lhd r$ can be substituted with =? to find the expected reward in any of the reward formulas discussed earlier. A satisfaction relational operator ($\models$) is used to define the semantics of CSL. The semantics are expressed in reference to states $s \in S$ and paths $\Psi \in Path^C(s)$ provided in a CTMC model. For instance, consider the following notations;

- $s \models \neg\Phi$: indicates that the state formula $\neg\Phi$ is satisfied in state s.
- $s \models a$: means that the atomic proposition a is satisfied in state s if $a \in L(s)$
- $s \models \Phi_1 \wedge \Phi_2$: states that the state formulas Φ_1 and Φ_2 are both satisfied in state s.

Example 2.7. In the following, we extend the software system presented in Example 2.3 to support verifying the system behaviour against requirements defined using CSL notations.

- $P_{\geq 0.85}[\neg fail\ U^{[0,T]}\ success]$: The probability that the system would produce produce accurate measurements within T seconds without the encounter of failure in sensor reading is $\geq 85\%$.
- $P_{\leq 0.95}[\neg success\ U\ fail]$: The probability that the system would produce more frequently inaccurate sensor readings before reaching for the first time a success state is $\leq 95\%$.
- $R^{reward}_{\geq 15}[C^{\leq T}]$: The expected amount of accurate reading accumulated up to reaching time T is more than 15.
- $R^{cost}_{\leq 32}[C^{\leq T}]$: The expected accumulated cost of energy to provide sensor reading up to time T is less than 32 energy units.

Analysing properties of a CTMC A path in a CTMC C is characterised as a *finite* or *infinite* path and it is used to evaluate properties. A *finite* path comprises of a bounded sequence $s_0 t_0 s_1 t_1 \ldots s_{n-1} t_{n-1} s_n$, where $s_0,\ s_1, \ldots,\ s_n \in S$. Since this is a *finite* path, the s_n needs to be an absorbing state. The transition rates for all $i = 0, 1, \ldots, n-1$ have to be $R(s_i,\ s_{i+1}) > 0$ and t_i is the time consumed while being in state s_i. In contrast, an *infinite* path is an endless sequence $s_0 t_0 s_1 t_1 \ldots$, where $s_0,\ s_1, \ldots \in S$, $t_0, t_1, \ldots$ correspond to the time spent in states $s_0, s_1, \ldots$ respectively. The transition rates for all $i = 0, 1, \ldots$ states in the sequence need to be $R(s_i,\ s_{i+1}) > 0$.

We further assume that the properties of the model C are analysed over a set of Paths^C which can combine several *finite* and *infinite* paths. For any given path ω, $\omega[i]$ denotes the i-th state on the path. If the given path ω comprises a *finite* sequence of states then the value domain of $i \in 1, 2, \ldots, n$, and if the path is an *infinite* path then $i \in \mathbb{N}_{>0}$. Furthermore, for a *finite* path we denote the time ($t(\omega, i)$), which represents the total time spent visiting the path states and $i \leq n$.

t_i corresponds to the time spent while being in the i-th state and $t \leq \sum_{i=0}^{n-1} t_i$. We use the notation $\omega@t$ to refer to the current state at time t. For an *infinite* path the time spent visiting path states $t(\omega, i) = \infty$ and $\omega@t$ refers to the state visited at t time.

2.5 Probabilistic Model Checking at Runtime

2.5.1 Description

Probabilistic model checking (PMC) is utilised for systems that exhibit stochastic behaviour to calculate the probability of specific condition including at runtime [153]. PMC provides a collection of techniques applied on a probabilistic Markov chain model to quantitatively verify the compliance of QoS properties to predetermined bounds [115]. The predefined QoS bounds are specified in a formal stochastic expression e.g. (Continuous Stochastic Logic **CSL** [9, 10], Probabilistic Computation Tree Logic **PCTL** [21, 91]). The probabilistic model represents the system behaviour as a finite state transition model. The transitions between these states are annotated with probabilities and the model can be extended to include cost/reward attributes. Two examples of properties obtained from [153] to illustrate probabilistic QoS properties in the field of fault-tolerant systems are as follows:

- ”the resource usage during the first month of operation should not exceed 100 units”;
- ”the likelihood of a failure occurring within the first hour is at most 0.001”.

The objective of probabilistic model checking is to verify the compliance of a given system model to certain constraints that are formulated using probabilistic temporal logic [117]. The verification requires a system model that captures the necessary environment‘s relevant assumptions. These assumptions illustrate the system’s desired results taking into consideration environment specification and requirements [193]. Particularly, PMC provides irrefutable evidence that the desired results can be achieved in view of domain knowledge representation. An example obtained from [35] further illustrates the use probabilistic model checking for quantitative verification. Assume that S and R represents the specification and requirements of a system. Additionally, let D formally capture the domain assumption of the same system. The desired requirements R hold if both the specification (S) and domain assumptions (D) are satisfied and consistently represent the system as:

$$S, D \models R \tag{2.8}$$

The equation indicates that domain assumptions are important factors for fulfilling the system’s desired requirements. This is because the satisfaction of requirements requires to reason about an upfront consistent representation of the target system. This representation captures certain assumptions about behaviour of the system to maintain expected requirements. If these assumptions, that describe the domain D, are inaccurate, then the fulfilment of the system requirements

is compromised. Calinescu et al. [35] used the term software evolution to describe the process of dealing with changes that divert the system from satisfying its requirements. They also suggest that a diversion may occur as a result of violations in any of the terms outlined in (2.8). After the software system is released, a system may require further evolution to cope with changes in system specification requirements or domain assumptions. The first change occurs when the system fails to deliver system functions as prescribed in its specification (S). The second change takes place when the devised domain assumption differs from actual environment behaviour (D). While, the third and final violation arises as consequence of misrepresenting system actual requirements (R).

The objective of our research work is to concentrate on changes influenced by the environment and detected by the violation of domain assumptions D. A change in captured domain assumptions indicate that a new domain property has been introduced. In turn, the newly introduced domain property requires modification to system specification S to conform with system requirements R [35]. Throughout this PhD project the term self-adaptive software system refers to class of systems that acquire autonomous capabilities to respond to changes in domain assumption D, in order to satisfy requirements R outlined in (2.8). The variability of domain assumptions is due to two main concerns.

The first concern is as a consequence of situating the software system in the environment where the behaviour is considered to be highly uncertain. The second concern is due to high degree of variance in environment behaviour that is likely to affect achieving desired requirements. To overcome these concerns, Calinescu and Kikuchi [38] argue that expression of system properties quantitatively permit acquiring irrefutable evidence that QoS requirements are met. Achieving QoS requirements (e.g., performance, efficiency, etc.) necessitate expressing these requirements with reference to system properties. Kwiatkowska [115] indicated that obtaining a mathematical model that reflects environment behaviour, in addition to, formal quantitative representation of the system properties, facilitates the comprehensive reasoning about achieving these properties to meet the system requirements. Therefore, these assumptions play a pivotal role in adapting system properties to satisfy QoS requirements.

Figure 2.15 illustrates a self-adaptive service based system that applies probabilistic model checking to satisfy QoS requirements as in formula (2.8) [35]. The example describes a self-adaptive medical assistance system that comprises of three main constituents. The first part contains a formal specification S of services provided by the system. The system initially receives a request that either require a further processing of the patient profile (service s2) or invocation of alarm service (s1) to notify the patient with the requested alarm. In case the request requires further analysis (s1), the outcome of this analysis directs the system to either changing the prescribed drug (service s3) alerting the patient using service s1 or by finalising the patient request. The second constituent, profiles the behaviour of the three services (s1,s2,s3) stated in system specification S. The behaviour assumptions D of these services are reported according to

the efficiency (prescribed cost) and robustness (prescribed failure probability) rates. While the final constituent formally states QoS requirements R based on system properties expressed in D. For example, the third requirement R3 states an efficiency threshold for handling requests.

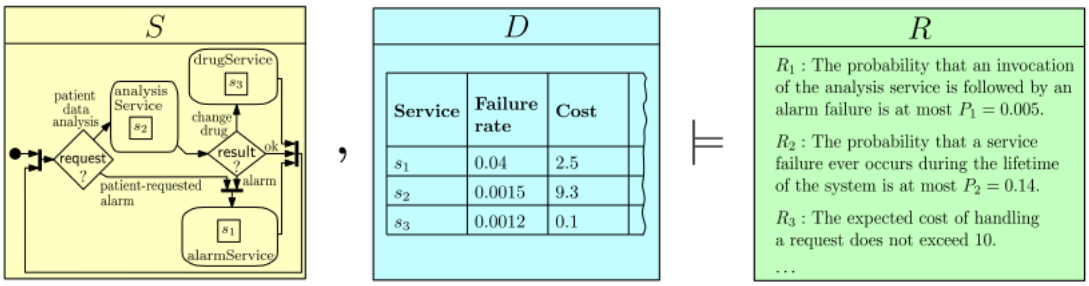

FIGURE 2.15. Application of probabilistic model checking in service based system [35].

The MAPE (Monitor, Analyse, Plan, Execute) control model maintains a centralised knowledge repository that captures an active system representation [64, 155]. This representation continuously preserves a model that combines environment behaviour assumptions D and system specification S. The monitor activity utilises *sensors* to constantly augment the system model representation with changes detected in the system itself or its environment. Subsequently, the analyse activity determines whether the change detected in monitor phase violates QoS requirements R. If a violation of requirements is detected, an adaptation plan is prepared by the plan activity. Then, the execute phase applies the adaptive change through the appropriate *effectors* [35, 105].

Quantitative verification technique can be utilised to mathematically verify system model representation conformance to QoS requirements [115]. The verification can be included in the analysis activity to conduct an exhaustive analysis to detect or even predict the QoS requirement violations. Requirement violation can be inferred from verification though various forms. A detected violation can be either a simple true/false feedback or a computation that yields a quantitative result. For instance, the verification outcome can possibly be a probability of service failure rate or even a cost prediction for handling incoming patient request as in 2.15 the model depicted in that represents a service based system.

Figure 2.16 illustrates quantitative verification technique to predict/detect requirement violation. The verification is conducted by utilising probabilistic model checking tools. MRMC [104] and PRISM [119] are considered the most frequently used tools. A list of alternative tools are referenced in [143, 145]. In the following section, we illustrate the latest progress on approaches that utilise runtime quantitative verification to analyse system compliance with adaptation requirements.

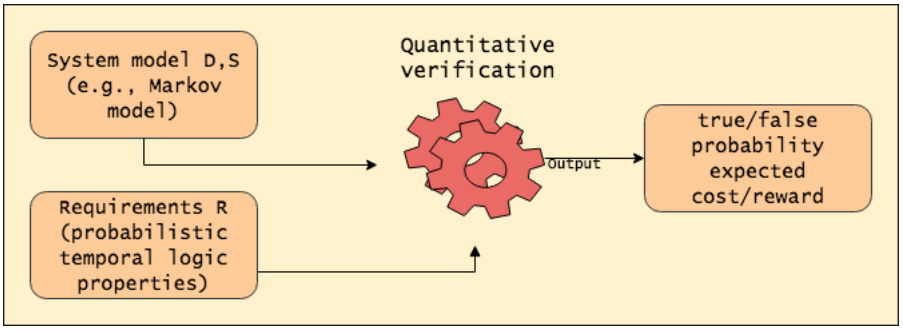

FIGURE 2.16. Quantitative verification using probabilistic model checking adopted from [35].

2.5.2 Frameworks

A self-adaptation framework has been proposed by Calinescu and Kwiatkowska [39] for integrating self-adaptive capabilities in legacy systems. They managed to utilise Markov chain models to represent system parameters and states at runtime. This representation allows to reason about the system using the PRISM probabilistic model checking tool [119]. They introduced an autonomic manager architecture that quantitatively verfies the system properties of interest to adapt the system behaviour according to the current system states and objectives. They define a set of essential three steps to make their autonomic architecture applicable to any legacy system that exhibits stochastic behaviour.

These steps are generation, deployment and exploitation and used to integrate and maintain the autonomic manager on legacy systems. These steps are divided based on the role of involved users (developer, system administrator, and end user). The approach was used to adapt the power consumption of a disk drive in reference to the system workload and policies. Similarly, they integrate the autonomic manager to adopt the availability of managed clusters in a data centre environment. The introduced framework managed to perform exhaustive quantitative analysis to reconfigure the system to meet the encoded probabilistic temporal logic policies. However, since the presented framework relies on centralised autonomic manager, the underlying system may experience delays in responding to adaptation concerns. Also the system resources such as memory and CPU may be affected by heavy workloads.

Epifani et al. [70] proposed KAMI (*Keep Alive Models with Implementations*) approach to adapt systems with distributed components at runtime. Changes presented by the execution environment demand a response to violations of the system‘s non-functional requirements. The KAMI approach utilises the Bayesian estimation technique [18] to update the state transition parameters in the system model (discrete time markov chain model). Updating the transition parameters facilitates the prediction of possible failures that could potentially affect the reliability of requirements. Also, the proposed approach reacts to adaptation concerns by using PRISM probabilistic model checking tool [119] to ensure that the system‘s non-functional requirements

are met.

Epifani et al. applied KAMI approach to Tele-Assistance (TA) medical system presented in [15] to evaluate their approach. The TA system architecture consists of distributed components that monitors a patient‘s vital health signs remotely and intervenes by sending First-Aid Squad (FAS) if the patient‘s health signs raise concerns. The system must conform to reliability requirements in which it has to reconfigure itself by means of adaptation to react to component failure concerns that may endanger the patient‘s life. The conducted testing shows that the self-adaptive system managed to predict possible failures through the application of Bayesian estimation technique. Also, their results indicate that the system adapts in the presence of violation detected by the PRISM model checking tool.

However, the proposed approach employs DTMC models that usually support system reliability requirements [89]. Therefore, the support of other non-functional requirements (e.g. performance and efficiency) necessitate extending the approach to support continuous time Markov chain models. Epifani et al. reported that the fulfilment of non-functional requirements highly depends on the precision in defining system parameters which are deduced at design time by domain experts. Also, the approach adapts to changes by altering the corresponding system parameters. Thus, the adaptation is restricted to changes in system parameters without further support to modify the model architecture.

Filieri et al. [73] proposed an extension to the KAMI [70] approach to support quantitative analysis of non-functional performance requirements. The analysis of performance requirements requires the expressing of system behaviour using continuous-time Markov Chain (CTMC) models. Unlike DTMC models, CTMC models take into consideration the execution time of each state in the transition path. Thus, capturing the time aspect in the system behaviour which facilitates the reasoning about time related requirements such as system throughput. Filieri et al. utilised CTMC models to analyse performance requirements and ensure that the execution of a certain process is within a required time boundary.

They expressed system performance requirements using continuous stochastic logic (CSL) [10] that supports the timing aspect. An e-commerce system proposed in [88] is used to validate the effectiveness of their approach. The utilised e-commerce system maintains a DTMC model to reason about reliability requirements and a CTMC model to verify the performance related requirements. Like the original KAMI approach, the extended ones supports predicting and detecting a violation of user predefined reliability requirements. However, the latter approach is extended to predict performance violations using the same Bayesian estimation technique [18].

Meanwhile, Filieri et al. conducted the quantitative verification using the mainstream probabilistic model checking tools. They have used PRISM [93, 116] and MRMC [104] checking tools to continuously verify that the system requirements are met. The results of the tests conducted by them indicate that there is a possible chance to encounter undesired delays in reacting to adaptation concerns. Therefore, they suggest the need to devise new strategies that can achieve

a more efficient and reliable adaptation. Efficiency and reliability in realising adaptation is essential to adapt the system to a variety of non-functional requirements. CTMC model for instance, annotates system states with cost and rewards that enable the verification of energy consumption requirements. At the same time, the model can be utilised to reason about another performance related requirement such as certain process latencies or throughputs. Also, finally, the proposed approach lacks the support of architectural change. According to Filieri et al., a more precision in understanding environment changes led to diversifying the architecture variety that can aid in realising a more effective adaptation at runtime.

Calinescu et al. proposed QoSMOS (*QoS Management and Optimisation of Service-based systems*) [37], a framework for realising self-adaptation in service based systems. This framework enables the development and management of a service based system that adapts in the presence of uncertainty in the system itself or in surrounding environment. The introduced framework utilises ProProST component to specify high-level QoS requirements which are required to reason about the system behaviour.

In contrast to previous approaches, QoSMOS framework provides a holistic solution that supports all MAPE adaptation activities. A component that implements KAMI approach [70] monitors the behaviour of the system and subsequently updates the corresponding system model with changes that occur in system and model parameters. The updated system model is then analysed by the PRISM [119] probabilistic model checking tool to verify QoS requirements obtained from the ProProST component as a set of probabilistic temporal logic formulas. The GPAC component [32] is then utilised to obtain the optimal configuration. The proposed framework is validated utilising the Tele- Assistance (TA) medical system defined in [15] work. The testing results reports concerns regarding the QoSMOS framework experiencing delays in reacting to adaptation. The experienced delay occurs as a consequence of employing the KAMI component which handles the monitoring and detection of requirement violations in a centralised manner. Calinescu et al. admit that the employment of KAMI component in large-scale systems may cause delays in responding to adaptation concerns.

Calinescu et al. introduced the DECIDE approach [34]. This is particularly relevant to the present PhD project as it is a self-adaptation approach that employs runtime quantitative verification in distributed systems with decentralised control. As such, we describe this approach in detail in Section 2.6.1.

In conclusion, runtime quantitative verification (RQV) provides rigorous assertion that a self-adaptive system complies to non-functional requirements. Probabilistic model checking tools can verify that continuously updated system models conform with predefined reliability, performance, and other non-functional constraints. The conducted survey that is summarised in Table 2.2, suggests that RQV approaches are less effective in large scale systems [37, 39, 70, 73]. The limitation in effectiveness is due to delays in detecting and reacting to adaptation concerns. Therefore, it is important to further examine efficient runtime verification approaches that reduce

the verification time and resource utilisation [35, 84]. However, decentralised control approaches such as DECIDE [34] managed to reduce resource overheads by dividing the computation burden between distributed and coordinating MAPE control loops. Weyns et al. [184] indicate that systems with decentralised control approach achieves better scalability in terms of computation and computation. Meanwhile, the DECIDE work shows that the adaptation requirements of a system component are prioritised over system global requirements. This issue may lead to sub-optimal adaptation decision from a system view point.

Table 2.2: Surveyed Self-adaptive systems that utilise Runtime Quantitative Verification

Proposed solution	Problem Scope	Application Domain	Control approach	Impact on QoS attributes	Limitation
Legacy system adaptation framework [39], 2009	Integrating autonomic manager to legacy systems	Dynamic power management in a Fujitsu disk disk drive [149]	Centralised control	Energy efficiency, Increase availability	Resource utilisation Overheads, Delay in reacting to concerns
KAMI self-adaptation approach [70], 2009.	Maintain and verify system models at runtime	Tele- Assistance (TA) medical system presented in [15]	Centralised control	Reliability (e.g. fault tolerance)	Support only reliability concerns, and adaptation actions that alter system parameters
Extended KAMI self-adaptation approach [73], 2012.	Maintain and verify system models at runtime	e-commerce system based on [88]	Centralised control	Reliability and performance	Architectural change adaptation, Delays in reacting to adaptation concern
QoSMOS Self-adaptation framework [37], 2011	Adaptation in service based system	Tele- Assistance (TA) medical system presented in [15]	Centralised control	Reliability and performance	Delays in reacting to adaptation concern, overheads in resource utilisation
DECIDE self-adaptation approach [34], 2015	effectiveness of RQV in distributed systems	Embedded systems deployed on distributed unmanned marine vehicles (UUV)	Decentralised control	Reliability and performance	Approach generalisation, confidence estimation in execution context

2.6 Distributed Self-Adaptive Systems with Decentralised Control

This section overviews a number of representative approaches to decentralising the control software of distributed self-adaptive systems. We start by describing in detail the approach that the project builds on (in Section 2.6.1, and then summarise other approaches relevant to the project (in Section 2.6.2).

2.6.1 The DECIDE Approach

The research presented in this thesis builds on the DECIDE approach to decentralising the control of self-adaptive systems introduced in [34]. As such, we will describe this approach in detail in this section.

DECIDE supports the development of SAS comprising collaborating components (e.g., robots, unmanned vehicles, or smart IoT devices). Formally, DECIDE SAS comprise $m > 1$ components. We use Cfg_i and Env_i to denote the set of possible configurations and the set of possible envi-

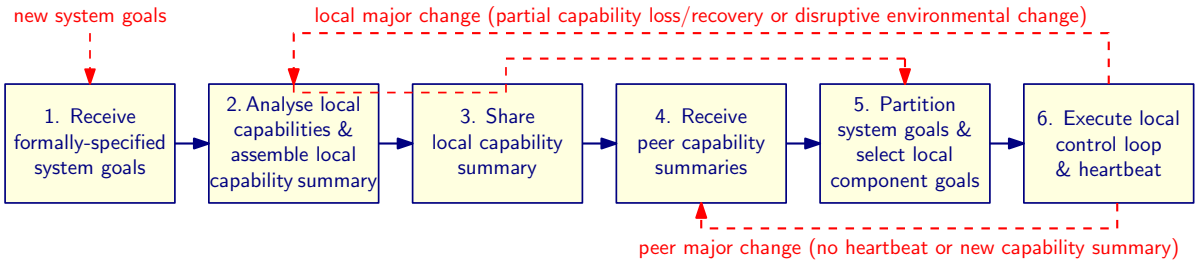

Figure 2.17: Decentralised control workflow executed by each component of a DECIDE distributed autonomous system; the dashed arrows indicate scenarios that require the re-execution of all or of a subset of the workflow steps

ronment states for the i-th component, respectively. Thus, Cfg_i corresponds to parameters of component i that can be modified, and Env_i to component parameters that can only be observed. In addition, component i has several functional and QoS *attributes*. Functional attributes may include the set of tasks that the component can perform for a given configuration, or the sequence in which it must execute certain tasks for specific environment states. QoS attributes are concerned with the reliability, performance, cost, etc. with which these tasks are executed.

Definition 2.15. An attribute of a DECIDE component with configuration space Cfg_i and environment space Env_i is a function $attr : Env_i \times Cfg_i \rightarrow V$ given by

$$attr(e,c) = f(e,c,pmc(M(e,c),\Phi)), \tag{2.9}$$

where:

- V is the set of all possible values for the attribute;
- f is a function that can be evaluated efficiently;
- $M(e,c)$ is a stochastic model that is typically component specific and may also be attribute specific;
- Φ is a probabilistic temporal logical formula over $M(e,c)$;
- pmc is a function that returns the result of the probabilistic model checking of formula Φ for model $M(e,c)$.

From the potentially numerous attributes and configuration/environment parameters of DAS components, DECIDE focuses on the attributes that appear in the system and local requirements, and on the parameters that influence the values of these attributes. Other component attributes and parameters are typically abstracted out. Two types of requirements are supported by DECIDE:

1. *Constraints*, which represent boolean-valued expressions that must be true, and are specified as first-order logic formulas over the component attributes;

2. *Optimisation objectives*, which represent real-valued expressions that must be minimised or maximised, and are defined as first-order logic terms over the component attributes.

System-level requirements (i.e. constraints and optimisation objectives) are defined over the attributes of all DAS components, whereas the local requirements of a component are defined over the attributes of that component. More specifically, a component-level constraint and a component-level optimisation objective have the form:

$$attr_k(e,c) \bowtie bound_k, \text{ where } \bowtie \in \{<, \leq, =, \geq, >\} \tag{2.10}$$

and

$$\text{minimise } attr_k(e,c) \text{ or } \text{maximise } attr_k(e,c) \tag{2.11}$$

respectively, for some attribute $attr_k$ of a component; while a system-level constraint and a system-level optimisation objective have the form:

$$\mathsf{OP}_{i=1}^{m}(attr_{i,k}(e,c)) \bowtie bound_k, \text{ where } \bowtie \in \{<, \leq, =, \geq, >\} \tag{2.12}$$

and

$$\text{minimise } \mathsf{OP}_{i=1}^{m}(w_i attr_{i,k}(e,c)) \text{ or } \text{maximise } \mathsf{OP}_{i=1}^{m}(w_i attr_{i,k}(e,c)), \tag{2.13}$$

respectively, where the operator OP applied to the k-th attribute of each of the m system components is one of $\sum_{i=1}^{m}$, $\prod_{i=1}^{m}$, $\bigvee_{i=1}^{m}$, etc, and $w_1, w_2, \ldots, w_m > 0$ are weights. Finally, to achieve autonomous operation, DECIDE systems and their components can have at most one optimisation objective each.

DECIDE supports distributed SAS that have the compute resources to perform the low-overhead control workflow shown in Fig. 2.17. This workflow has six steps and is executed locally by the instance of the decentralised DECIDE controller running on each SAS component.

In step 1, the component receives a copy of the system goals, specified formally as described later. This step is performed infrequently—when the component joins the system and, if applicable, when new system goals are provided by its owners.

In step 2, probabilistic model checking is used to analyse stochastic models of the component and its environment. The purpose of this analysis is to formally establish the component capabilities that are relevant to the system goals (e.g., throughput, response time and cost). This analysis takes into account any component-level requirements that might exist, and its result is a *local capability summary*, i.e., a finite set of alternative contributions that the component is *guaranteed to be able to make* towards achieving the system goals, for the assumed range of environmental scenarios. These alternative contributions correspond to different levels of quality of service (e.g., latency, reliability or energy consumption) that the component can achieve. Each contribution is encoded as a tuple that specifies the attribute values for that contribution:

$$(attrValue_1, attrValue_2, \ldots, attrValue_n), \tag{2.14}$$

where this tuple only includes the values for the $n > 0$ attributes that appear in the system-level requirements (2.12) and (2.13). This step and the sharing of the local capability summary with the peer SAS components (e.g., through a broadcast) in step 3 are also executed infrequently—when the component joins the SAS, after receipt of new system goals, and when the component experiences partial capability loss/recovery or disruptive environmental changes.

In step 4, the DECIDE controller instance receives similar capability summaries computed by peer SAS components as they join the system or experience *local major changes* (i.e., changes that require a re-partitioning of the system goals, and that are local to a specific component). As indicated in Fig. 2.17, this infrequent step is skipped when the assembly and sharing of a new local capability summary in steps 2 and 3 are due to a local major change (because the capabilities of the peer components are unaffected in this case).

Each analysis of the local capabilities (when joining the system, when the system goals change, or after local major changes) and the receipt of new capability summaries from the peer components trigger the (re-)partition of the system goals among the available components and the selection of local component goals. These tasks are executed in step 5.[1] An efficient, deterministic mathematical programming technique is used by the DECIDE controller of every component to produce an identical partition of the system goals and the rigorous selection of local goals on each SAS component. This replication of the system-goal partition on every component ensures that the system does not have a single point of failure (and reduces its communication needs) with only modest additional computation overheads. The selected local component goals represent one of the alternative contributions from the capability summary of the local component, chosen such that *the system is guaranteed to comply with its goals* as long as each component achieves its local goals.

Most of the time, the only activities performed by DECIDE are the execution of a local control loop and the maintenance of a low-overhead system "heartbeat" in step 6. The local control loop ensures guaranteed compliance with the selected local component goals through local adaptation underpinned by the probabilistic model checking of continually updated stochastic models of the component and its environment. Infrequently, events such as disruptive environmental changes (e.g., significant workload increases) or loss of component capabilities (e.g., the loss of a sensor) render a DECIDE local control loop unable to achieve the local goals; or the disappearance of a heartbeat indicates the complete failure of a peer component. These events (shown by dashed arrows in Fig. 2.17) are termed major changes, and their detection by the local control loop is followed by the computation and selection of new sets of local goals for the (remaining) SAS components.

[1] To avoid re-executing step 5 many times when the SAS is first formed (or when the system goals change) and multiple components share their capabilities summaries at slightly different times, DECIDE uses batch processing to handle new capability summaries. To this end, the computation of a new local summary or the receipt of a peer summary starts a small time window, and all new summaries computed or received within this time window are processed together by executing step 5 only once, at the end of the window.

As implied by the description so far, DECIDE is applicable to SAS that present the following characteristics:

1. The system goals and the component-level requirements can be analysed through the probabilistic model checking of stochastic models of the component behaviour and the environment. Recent research on self-adaptive systems [114] shows that this characteristic is common to a broad enough range of service-based [35], robotic [96], cyber-physical [22], IoT [137] and cloud computing [35] systems to make DECIDE widely applicable.

2. The average time interval between successive major changes within the entire system is much larger than the time required to execute steps 1 to 6 of the DECIDE workflow (e.g., at least two orders of magnitude larger, or many minutes versus a few seconds). A key distinguishing feature of DECIDE is its use of a *tolerance factor* that allows a wide range of SAS to be augmented with this characteristic. This tolerance factor quantifies the level of environmental uncertainty considered in the local capability analysis from step 2 of the DECIDE workflow. In particular, a large tolerance factor makes this analysis very conservative, yielding local capability summaries with alternative contributions that SAS components can easily achieve through local adaptations at almost all times. This can greatly reduce the frequency of major changes, albeit at the cost of operating with highly suboptimal contributions from the SAS components.

3. The communication between SAS components—using broadcasts, multicasts or at least peer-to-peer messaging—is infrequent and modest in size (e.g., tens of bytes per hour) and can cope with high latency, but needs to be reliable. Given the availability of middleware that can provide reliable communication over unreliable channels [106], many SAS can be engineered to exhibit this characteristic.

Limitations of the DECIDE approach. While the philosophy underpinning the DECIDE approach (i.e., the workflow from Figure 2.17) is general, its instantiation from [34] has major limitations:

1. The set of local capability summaries exchanged by DECIDE components represent *alternative* contributions that a component can make, and the partitioning of the system goals needs to select one and only one of the summaries from this set. This significantly limits the applicability of DECIDE, e.g., precluding the use of the approach in distributed SAS applications comprising heterogeneous tasks or tasks whose execution can be split into parts assigned to different components.

2. The combinations of system-level constraints and optimisation objective that can be handled by DECIDE are limited to those that can be formalised as a multiple-choice knapsack

problem [161]. While this is an effective mathematical programming technique, it can only encode a small range of goals of real-world distributed SAS applications.

3. The proof-of-concept implementation of DECIDE from [34] is application specific and cannot be used for any other distributed SAS than the simulated unmanned underwater vehicle (UUV) team it was implemented for.

4. The evaluation of the DECIDE approach is also limited to the UUV domain for which an implementation was available, and it was only done in simulation.

The contributions of these thesis address all of the above limitations of DECIDE in Chapters 3, 4 and 5.

2.6.2 Other Approaches

Claes at al. [51] proposed a self-assembly approach tailored towards swarm robotics applications. The author introduces a simulation for a warehouse environment using a virtual environment where there are a set of tasks that need to be distributed to teams of robots. The approach addresses the task allocation problem using decentralized planning where each robot solves a Multi-Agent Markov Decision Process problem by modeling other robots using a Statistical approach based on Monte Carlo Tree Search (MCTS). Each robot maintains a fixed computation budget to perform planning at run-time to adapt to unexpected changes in the environment.

Matena at al. [131] demonstrates a model problem comprising of a number of autonomous robots embarking on an office cleaning mission. The model problem includes a test-bed to facilitate experimentation of self-adaptation techniques. Both the robots and the model problem are implemented using ROS (Robotics Operating System) which provides a standard set of libraries and services used in robotics implementation. The provided test-bed permits experimenting adaptation techniques by using ROS-based stage simulator or deployment on an actual Turtlebot robots. The provided adaptation logic is specified using DEECo, which is a compnent model based on concepts of ensemble-based component systems (EBCS). The adaptation logic supports the decentralised cooperation of software components at runtime.

FlashMob [167] is a decentralised control solution that implements the self-assembly adaptive property to guide nodes' assembly and configuration in a network. The approach is based on the three-layer model for autonomous systems and focuses on qualitative aspects such as fault-tolerance. The self-assembly and configuration are guided by simple utility functions that encode the non-functional preferences. The consensus among the nodes is achieved using a gossip protocol. The nodes in such a network are unaware of the full solution space, so the protocol facilitates self-assembly and configuration.

Brun et al. [27] present an architectural style for solving computationally intensive problems across large networks in a distributed environment. The presented architecture ensures fault-tolerance, scalability, and discreetness in large networks. This study's unique feature is that

it is the first method that combines architectures and self-assembly. The study declares that the proposed method can solve all NP-complete problems by demonstrating its applicability to problems [25] and [28]. However, they lack the demonstration of problems involving probabilistic quantitative guarantees, and the method is unable to overcome uncertainties.

In [94], the authors present a case study defined using a framework for specifying autonomic components and the cooperation strategies among these components. The framework supports the rapid experimentation and reasoning about the different coordination strategies using a discrete-event based simulation framework called IVIS. The use-case used for this proposed method is from smart farming domain and based on actual scenario taken from AFarCloud project [44]. The research focuses on how complex collaboration rules are quickly captured by autonomic components (drones) and non-controllable environment agents (flocks of birds in their example). Both discrete event simulation and real-time visualization of the use case are described. The cooperation constructs are defined using an Internal Scala-based DSL and evaluation system for dynamic groups at runtime.

A multi-agent approach is described in [59] for balancing performance and power in data centres using specialized cloud environments for performance/power balancing. They use the strategy of self-adaptive managers for sharing of information and synchronization. Unity, a decentralised architecture for self-managing distributed computing systems enabling self-configuration at initialisation and self-optimisation at runtime is introduced by [59], [170]. Every system component in Unity consists of autonomic element that can control its own resources for meeting their individual requirements as well as delivering services to other autonomic elements. Global view of demand and availability of resources are maintained by resource arbiter and is responsible for allocating the resources to autonomic elements after evaluation of service-level utility function.

In [112], [108] and [179], authors propose approaches to establish self-organisation in systems consisting of distributed agents. The aim is to create coalitions between agents dynamically to fulfil a system objective. In [112], the organisation's performance is improved using the information that is locally maintained in each agent. To improve performance, agents communicate regularly in groups to decide actions that enhance the organisation's performance. Kim et al. [108] introduce a framework that varies the degrees of cooperation between agents. A robotic system is implemented that includes robots that are organised in coalitions. Each robot communicates with a subset of its peers to determine the degree of cooperation between the coalition members. Robots address adaptation concerns such as failures or lack of resources and communication by adjusting the level of autonomy and collaboration among the coalition members.

Decentralized planning is described in MAPmAKER [133], where multi-robot systems are the primary target. MAPmAKER distributes the robots in different classes considering the local mission of each robot. A variant of a three-valued LTL semantics planner is used to reason under partial knowledge about mission satisfaction. However, the proposed approach is limited to reason about the functionality aspects of the system and does not consider extra-functional

concerns.

The work by Cheng et al. [50] proposes a hypothetical news service system called *Znn.com*. The *Znn.com* system provides news content in variable forms (e.g. multimedia, text), depending on the constraints such as response time and budget limit. These constraints influence the provisioning and discharging of available servers and contribute towards altering the response (*fidelity*) by changing the delivered content type (e.g. from multimedia to textual news) to serve the end users with customised news content. The system implements a master/slave control approach based on the Rainbow framework [79]. Since the news service components are distributed across multiple servers, the system has to conform to the Rainbow framework by implementing a set of probes and gauges to report system status to the centralised *architecture evaluator*. In turn, the architecture evaluator analyses the received data to detect any anomalies in response time. If a problem is detected, an *adaptation manager* intervenes to decide on the optimal strategy to restore the response time to the required threshold. Finally, the decided strategy is executed by the *strategy executor* utilising the distributed effectors.

Cheng et al argue that Rainbow framework is applicable to various domains because of the analyse and plan parts of the control being centralised. To apply the framework to a target system as in *Znn.com* system example, a developer has to customise the *model manager* to reflect the architecture model of that system. Similar customisation is required in the architecture evaluator and the strategy executor to project the business objectives and strategy effectors on the target system. The *Znn.com* system example indicates that complying to master/slave control approach (Rainbow framework), increases the reusability to a wide range of applications. However, centralising some parts of the control activities may result in communication overheads. Particularly, if the number of system components that implement probes and gauges increase. Also if the centralised parts of control fail, the adaptation property becomes unattainable. Other examples that illustrate the Master/Slave control approach can be found in [77, 172].

CHAPTER 3

Mathematical Programming Techniques for Goal Partitioning in Self-Adaptive Systems with Decentralised Control

This chapter presents a repertoire of system-level goal partitioning techniques that enable the applicability of SAS control decentralisation philosophy presented in Section 2.6.1 to a broad range of distributed systems. This repertoire generalises the DECIDE solution proposed in [34], and therefore it underpins a new SAS control decentralisation approach that we call nuDECIDE.

We start the chapter by introducing, in Section 3.1, the generalised format of the system constraints and optimisation objective supported by nuDECIDE. We then present, in Section 3.2, the use of exhaustive search to identify the optimal partition of the system goals among the components of a distributed SAS. This baseline technique is not scalable, and therefore can only be employed when the number of possible goal partitions is relatively small. Next, Sections 3.3 and 3.4 describe the use of linear programming and integer programming for the efficient partition of the system goals, respectively, and we explain the prerequisites that the distributed SAS requirements must satisfy for these techniques to be applicable. Finally, in Section 3.5 we introduce the use of MDP policy synthesis as an additional, sophisticated technique that can be used to achieve the goal partition for applications whose nondeterminism cannot be easily expressed as an linear or integer programming problem (e.g., where task executions are not guaranteed to succeed). We conclude the chapter with a brief summary that overviews the merits and limitations of each of the proposed techniques in Section 3.6.

3.1 nuDECIDE system requirements

nuDECIDE generalises the system-level constraints and optimisation objective of DECIDE by relaxing the assumption that each of the system-level constraints (2.12) and optimisation objective (2.13) depends on a single attribute $attr_{i,k}$ of the i-th system component. Removing this assumption greatly increases the types of requirements that our approach can handle, and the goal partitioning problem to be solved takes the form detailed below.

Given a *sequence* of $k_i > 0$ possible contributions

$$\langle c_{i,1}, c_{i,2}, \ldots, c_{i,k_i} \rangle$$

of form (2.14) for each system component $i = 1,2,\ldots,m$, find the *amounts* $x_{i,1}, x_{i,2}, \ldots, x_{i,k_i} \geq 0$ *of each of these* k_i *contributions* that component i should be assigned such that *generalised* system-level constraints and a system-level optimisation objective of the form:

$$f_j\left(\left\{c_{i,1}, c_{i,2}, \ldots, c_{i,k_i}\right\}_{i=1}^{m}, \left\{x_{i,1}, x_{i,2}, \ldots, x_{i,k_i}\right\}_{i=1}^{m}\right) \bowtie_j bound_j, \tag{3.1}$$

where $j = 1,2,\ldots,n$, $\bowtie_j \in \{<, \leq, =, \geq, >\}$, and

$$\text{minimise/maximise } g\left(\left\{c_{i,1}, c_{i,2}, \ldots, c_{i,k_i}\right\}_{i=1}^{m}, \left\{x_{i,1}, x_{i,2}, \ldots, x_{i,k_i}\right\}_{i=1}^{m}\right), \tag{3.2}$$

respectively, where f and g are functions that can take any form.

Note that this generalised formulation replaces:

- the DECIDE use of a set of component contributions, out of which the goal partitioning selects one and only one contribution per component (see Section 2.6.1) with a sequence of contributions, each of which can be selected in a certain "amount";
- the operator OP from system-level requirements (2.12) and (2.13) with generic functions f and g.

Both of these generalisations greatly expand the types of problems that nuDECIDE can handle by comparison to DECIDE. In particular, the problem that can be solved by DECIDE is the very special case of this general problem in which

$$\forall i \in \{1,2,\ldots,m\}.\left(\exists j \in \{1,2,\ldots,k_i\}.x_{i,j} = 1 \wedge \left(\forall j' \{1,2,\ldots,k_i\} \setminus \{j\}.x_{i,j'} = 0\right)\right).$$

3.2 Goal partitioning using exhaustive search

3.2.1 Theoretical foundation

While exhaustive search does not come across as an attractive solution for the dynamic partitioning of the system goals, the technique has been used successfully for the selection of new configurations in self-adaptive systems using runtime quantitative verification [39]. In this section, we extend the use of exhaustive search to the goal partitioning problem from Section 2.6.1.

For the generalised goal partition problem from the previous section to be solvable by exhaustive search, we make the additional assumption that each element x_{ij} of the solution can only take a finite number of values or, equivalently, that the set

$$\{(x_{1,1}, x_{1,2}, \ldots, x_{1,k_1}, \ldots, x_{m,1}, x_{m,2}, \ldots, x_{m,k_m}) \mid \\ \forall j = 1,2,\ldots,n. f_j\left(\{c_{i,1}, c_{i,2}, \ldots, c_{i,k_i}\}_{i=1}^{m}, \{x_{i,1}, x_{i,2}, \ldots, x_{i,k_i}\}_{i=1}^{m}\right) \bowtie_j\}$$

is finite and has a reasonably small number of elements.

When this is the case, we assume that the result of the goals allocation problem can be represented as a matrix (X) of size $m \times \max_{i=1}^{m} k_i$. The matrix describes the results of goal partition problem by identifying the finite amounts $x_{i,1}, x_{i,2}, \ldots, x_{i,k_i} \geq 0$ of each of these k_i contributions for each system component $i = 1,2,\ldots,m$. In particular, the indexes $[x_{i,k}]$ of the matrix refer to i-th component and the amounts of each of these k_i contributions that component i should be assigned such that system-level constraints and system-level optimisation objective are satisfied. An element in the $m \times \max_{i=1}^{m} k_i$ matrix X, is denoted by $x_{i,k}$, where i and k vary from 1 to m and k_i, respectively. For each row $i = 1,2,\ldots,m$ in the matrix if $k_i < \max_{i=1}^{m} k_i$, padding with zero is used to substitute the missing contribution for the i-th component. Thus, if the value assigned to the matrix element $x_{i,k}$ is 0, then the i-th component either does not to perform any amount of the k-th contribution, or does not have the required capability for such a contribution (e.g., if the contribution corresponds to a task for with the component lacks the necessary actuator).

3.2.2 Example

Consider a distributed system comprising m mobile robots that are required to take periodic measurements of some aspect of their environment (e.g., level of lighting), so that measurements at N different locations are taken within t time units. Further assume that the robot travel time between locations is negligible compared to the time required to perform the actual measurements, that the mobile robots are provided with a reliable collision-avoidance mechanism, and that robot i, $1 \leq i \leq m$ has $k_i \geq 1$ modes of operation. Each operation mode is associated with different performance/energy use trade-offs, so that when robot i operates in mode $1 \leq k \leq k_i$, it requires t_{ik} time units and consumes e_{ik} energy to perform one measurement. As such, the local contribution summary from robot i has the form

$$\langle (t_{i,1}, e_{i,1}), (t_{i,2}, e_{i,2}), \ldots, (t_{i,k_i}, e_{i,k_i}) \rangle$$

The goal partitioning problem is to allocate to robot i $x_{i,1}$, $x_{i,2}$, ..., x_{i,k_i} measurements per t time units *and* operating modes 1 to k_i, such that the system accomplishes its mission:

1. The constraint set C comprises the following constraints:

$$C_1: \quad \sum_{i=1}^{m} \sum_{k=1}^{k_i} x_{i,k} = N,$$

$$C_2: \qquad \forall\ 1 \le i \le m\,.\,\#\left\{x \in \left\{x_{i,1}, x_{i,2}, \ldots, x_{i,k_i}\right\} \mid x \neq 0\right\} = 1,$$

$$C_3: \qquad \forall\ 1 \le i \le m\,.\,\sum_{k=1}^{k_i} x_{i,k}\, t_{i,k} \le t.$$

where constraint C_2 specifies that a single mode of operation is to be used for each robot (i.e., only one of $x_{i,1}, x_{i,2}, \ldots, x_{i,k_i}$ can be non-zero).

2. The optimisation objective

$$O: \qquad \text{miminise} \sum_{i=1}^{m} \sum_{k=1}^{k_i} x_{i,k}\, e_{i,k}$$

subject to the constraints C_1, C_2 and C_3 being satisfied.

To give a numerical example, consider that the system has $m- = 3$ robots with the following modes of operation:

- Robot 1 has $k_1 = 2$ modes of operation with $(t_{11}, e_{11}) = (17, 5)$ and $(t_{12}, e_{12}) = (30, 7)$.
- Robot 2 has only $k_2 = 1$ mode of operation with $(t_{21}, e_{21}) = (14, 5)$.
- Robot 3 has only $k_3 = 1$ mode of operation with $(t_{31}, e_{31}) = (18, 5)$.

Also, consider that the number of locations is $N = 7$ and that the measurements at these locations need to be carried out within $t = 30$ time units.

In this simple example, the problem can be easily solved through exhaustively analysing all possible measurement (i.e., goal) partitions, as summarised below.

nuDECIDE approach relies on formal methods that enable the analysis of the behaviour of the robot and its environment. Namely the approach employs probabilistic model checking, which are referred to in Chapter 4, to identify alternative contributions that are characterised by the system constraints and objective. These contributions are perspective about what a robot can offer towards the fulfilment of system goals, and what are the quality-of-service (QoS) characteristics of these contributions. Given a system with the mission characteristics described earlier, the m robot systems need to identify their possible contributions that are characterised by system constraints and objective. We further assume that robot i identifies and shares its contribution summary per its modes of operation $1 \le k \le k_i$. These contribution summaries have to satisfy robots local constraints and identified using a local optimisation function. Depending on the optimisation objective criteria (e.g, whether to maximise or minimise some optimisation criteria), the function selects a summary of contribution per each operating modes of robot i.

Since the size of the variables and constraints of the introduced problem in this example is relatively small, we may solve the problem using the brute-force search approach. The following

allocation matrix $[x_{i,k}]$ illustrates the optimal assignment of measurements (tasks) to robots.

$$X = \begin{bmatrix} 4 & 0 \\ 2 & 0 \\ 1 & 0 \end{bmatrix},$$

where $[x_{i,k}]$ identifies the finite the amount of measurements $x_{i,1}, x_{i,2}, \ldots, x_{i,k_i} \geq 0$ of each of these k_i contributions for each system component $i = 1, 2, \ldots, m$. For instance, robot 1 has to perform sensor measurements at ($N = 4$) locations using its ($k_1 = 1$) modes of operation. Moreover, the former assignment matrix $[x_{i,k}]$ can be rewritten to illustrate the allocation of measurements but in terms of the upper bound energy cost at which the i-th robot needs to perform its measurement per its operation modes $k_i \geq 1$. In particular, the matrix $x'_{i,k}$ identifies the finite amount of energy $e_{i,1}, e_{i,2}, \ldots, e_{i,k_i} \geq 0$ of each of these k_i contributions for each system component $i = 1, 2, \ldots, m$.

$$X' = \begin{bmatrix} 28 & 0 \\ 28 & 0 \\ 18 & 0 \end{bmatrix}$$

The matrix $x'_{i,k}$ defines the total energy that robot i can consume during the constrained mission time (t=30 minutes) to execute its assigned task(s). Consider for example robot ($i = 1$), according to the assignment matrix, robot ($i = 1$) has to fulfill its partition of system goals by executing the sensor reading task at 4 locations. Consider for example robot ($i = 1$), the robot needs to perform its measurements and consume energy less than 28 joules.

3.3 Goal partitioning using linear programming

3.3.1 Theoretical foundation

Linear programming (**LP**) is a mathematical technique dedicated for problems that can be mathematically formulated using linear equations and inequalities to find optimal solutions. If a problem can be formulated as a mathematical equations of a linear program, the method is used to find the best solution to the problem. Linear programming problems include a set of constraints that take the form of linear inequality and a linear objective function. The problem variables are often termed *decision variable* as the software solver need to determine the set of quantities that solve the problem. When the objective function which is typically encode some optimisation metrics such as utility, time and energy, yields an optimal value, the values of decision variables used to maximise/minimise these metrics are optimal and the optimisation problem is considered to be *solved*.

The mathematical method can be utilised to find optimal solutions for class of problems that deal with allocation of scarce resources to competing activities [173]. Such problems can be expressed using set of linear decision variables and constraints that define the boundary on the values of the variables. An optimal solution to a problem is obtained by a linear objective function

that measures and rank the contribution of each variable to the desired outcome. Depending on the desired solution whether to maximise a gain or minimise a cost, a linear objective function yields the optimal (largest or smallest) set of values for the problem variables that meet the problem constraints. Many real-world problems can be modeled and solved as a linear program problem. The process of solving these problems require finding the adequate formalism for the problem, which is a non trivial task [45]. Once an adequate formalism is arranged, a software solver can be used to obtain the answer to the problem.

To exploit linear programming for the partitioning of the system goals within the nuDECIDE approach, the constraints f_j and optimisation objective g from (3.1) and (3.2), respectively, must be of the appropriate form. Specifically, all the equations and inequalities used to formulate the constraints f_j and optimisation objective g should be of a linear form as follows.

$$f_j: \quad \sum_{i=1}^{m}\sum_{k=1}^{k_i} x_{i,k}F_j(i,k,c_{i,k}) \leq bound_j\left(\left\{\langle c_{1,1},\ldots,c_{1,k_1},\ldots,c_{m,1},\ldots,c_{m,k_m}\rangle\right\}_{i=1}^{m}\right), \tag{3.3}$$

where:

- $F_j(i,k,c_{i,k}) \in \mathbb{R}_{\geq 0}$ is a function that takes the k-th contribution (2.14) of component i and returns a non-negative value based on the attribute values within this contribution;
- $bound_j\left(\left\{\langle c_{1,1},\ldots,c_{1,k_1},\ldots,c_{m,1},\ldots,c_{m,k_m}\rangle\right\}_{i=1}^{m}\right)$ is a function that returns a non-negative value based on the contribution summaries of all components.

$$\text{minimise/maximise } g: \quad \sum_{i=1}^{m}\sum_{k=1}^{k_i} u_{i,k}x_{i,k} \tag{3.4}$$

where $u_{i,k} \geq 0$ is a "utility" associated with component i and the k-th tuple from the contribution summary of component i. These utilities are coming from the system-level requirements (i.e., the system-level optimisation objective).

In general, throughout the context of our project, decision variables are used to quantify a contribution associated with a system component. This interpretation of variables can be used to formulate many types of allocation problems. For instance, variables can be defined to measure the amount of resource to use in a service based system and cloud system. The problem model presented in [178] can be extended to support self-adaptation with decentralised control in the domain of service-based systems. Thus, the use of nuDECIDE platform is not limited to certain domain and applications, but can be extend to support self-adaptation in areas that require a decentralised control and can be modeled and analysed using probabilistic model checking. Moreover, nuDECIDE platform can be applied to class of systems that exhibit system-level requirements, which can be formulated as linear programming problem. For any problem formulation, the most important initial step is to define the variables of the problem [80]. A defining of variables play essential role in reducing the size of the problem and transforming a problem from nonlinear to other linear variant [20]. In particular, throughout this thesis we consider linear

programming problems in standard form, in which the variables are all non-negative. However, there is a transformation technique [60] to cast a problem with unrestricted variables to an equivalent standard form problem with non-negativity constraints. Such techniques are beyond the scope of this thesis, as variables are utilised to indicate the amount of a resource consumed or the level of a certain activity. The following section illustrates the process of formulating the linear problem model for the plant inspection case study. The process description provides a more concrete idea of classes of allocation problems that can be supported in nuDECIDE platform.

Many real-world problems that require optimization tasks can be formulated as a linear programming problem. Given that the constraints and objectives of these problems can be expressed in linear functions. This section describes the necessary steps to translate system-level requirements of a candidate nuDECIDE application into mathematical equations that conform to linear program model. The steps description overlooks the question of how LP problems are solved, instead focuses on how problems are formulated to linear programs, while a computer solver provides the solution to the formulated problem.

3.3.2 Example

In this section, we illustrate the application of the theoretical results from Section 3.3.1 to a system comprising $m = 3$ mobile robots required to take regular measurements to monitor the possible leakage of hazardous gases in an oil refinery. This section provides the description of this system, and maps its associated goal partitioning problem to a linear programming problem. Later in Chapter 5, we will present a complete implementation and evaluation of this case study, using three mobile robots in a testbed that emulates the real system.

We assume that these measurements need to be taken along $L_1 > 0$ metres of methane gas pipelines, and $L_2 > 0$ metres of propane gas pipelines, and that they need to be completed within a duty rounds of T_{round} time units. Each robot $i,\ 1 \leq i \leq m$ has $m_i \in \{2,\ 4\}$ sensors, i.e., (i) a pair of sensors ($m_1 = 2$) specialised to execute an inspection task of $type = taskTyp_m$; or (ii) a pair of sensors ($m_2 = 2$) specialised to execute an inspection task of $type = taskTyp_p$; or (iii) a pair of each type of sensors (when $m_3 = 4$). The first type of tasks ($taskTyp_m$) requires the robot to use its sensors to measure the presence of methane gas, while the second type ($taskTyp_p$) entails that the robot checks for the presence of propane gas leak.

A measurement taken by a sensor is associated with different measurement time/energy use trade-offs, so that when robot i uses sensor $1 \leq j \leq m_i$, it requires t_j time units and consumes e_j energy to perform one measurement. Figure 3.1 illustrates a DTMC model that describe the behaviour of robot i. In the initial state (s_0) of this model, the robot takes one reading of the environment characteristic it is measuring. This reading is using sensor j_1 with probability p_i (modelled by a transition to state s_1) or sensor j_2 with probability $1 - p_i$ (modelled by a transition to state s_2).

A sensor reading taken by sensor j_1 is of insufficient accuracy with probability p_1^{retry} (mod-

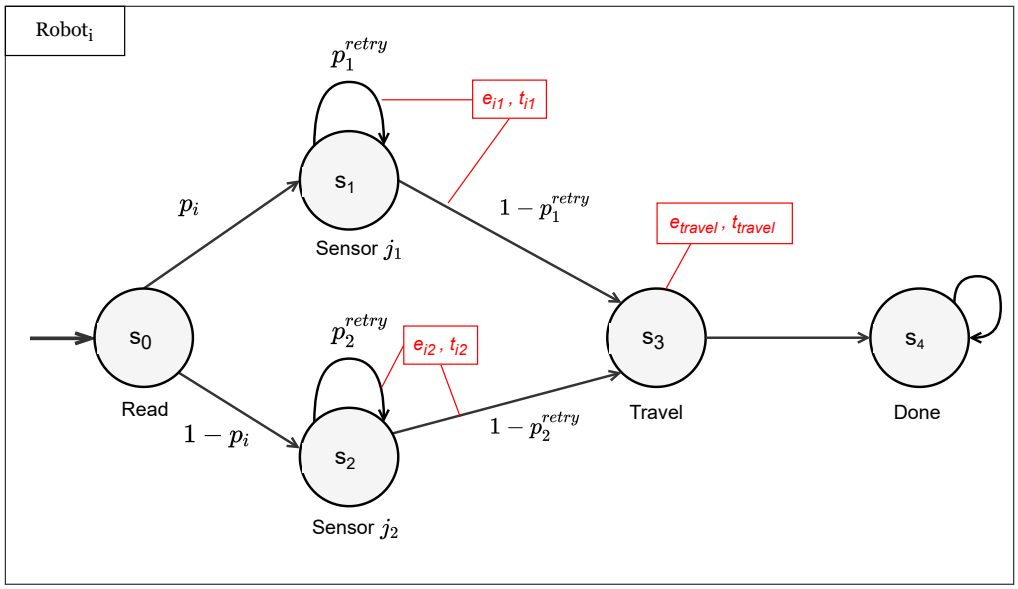

Figure 3.1: DTMC model of the i-th robot

elled by a transition to state s_1) or successful with probability $1 - p_1^{retry}$ (modelled by a transition to state s_3). Similarly, a sensor reading taken by sensor j_2 is of insufficient accuracy with probability p_2^{retry} (modelled by a transition to state s_2) or successful with probability $1 - p_2^{retry}$ (modelled by a transition to state s_3). In state (s_3), the robot travels one meter of the assigned segment of L_1 or L_2, which is modelled by a transition to the final state s_4. The DTMC is augmented with two reward structures, an "energy use" reward structure, and a "time" reward structure. The "energy use" and "time" reward structures determines the energy cost associated with using any of the robot's sensors in taking one reading of the environment characteristic and traveling one meter of the assigned segment.

We further assume that robot i moves between the task locations with speed $sp_i \in (0,\ sp_i^{max}]$ and the travel time between locations is captured by the reward structures of the DTMC model shown in Figure 3.1, which illustrates the abstract behavior of the i-th robot in carrying out an inspection task using its j-th sensor. The j-th sensor of robot i operates with probability p_j^{retry} that one of its readings is not of adequate accuracy for the mission purpose. The probability captures the dynamics of the environment, in which a sensor may fail or face degradation in its performance. Also, the robot speed (sp_i) may affect the accuracy of its sensor readings. A successful completion of the mission requires the collaboration among the team of robot systems to meet the system-level requirements from Table 3.1. Along with these system-level requirements, each robot i must satisfy its local requirements from Table 3.2.

The DTMC model shown in Figure 3.1 is used by robot i to identify the cost associated with contributing to system-level requirements. In particular, robot i computes the energy and time associated with using its sensor to take a one measurement, and moving a certain distance. Also,

Table 3.1: System-level requirements of multi-robots surveillance mission

Requirement	Description	Type
R1	The m- robots should take 1 measurement per meter of the assigned segment of L_1 and/or L_2	Constraint
R2	The m-robots must complete the assigned tasks within T_{round} minutes of the time available to complete a duty round	Constraint
R3	Subject to satisfying R1 and R2 the multi-robot system should maximise the utility of the mission, where the utility of covering one metre of methane pipeline is $u_1 > 0$ and the utility of covering one metre of propane pipeline is $u_2 > 0$. (We assume that any area not covered by the robots is covered by a human operator, and the aim of the optimisation objective is to minimise this remaining work.)	Objective

Table 3.2: Component-level requirements of multi-robots surveillance mission

Requirement	Description	Type
R4	The energy required by robot to survey the assigned area should not exceed E_i^{max} Joules per duty round of mission	Constraint
R5	The time required by robot to complete the assigned area should not exceed t_i^{max} minutes per duty round of mission	Constraint
R6	If multiple configurations satisfy R4 and R5, then the i-th robot should use the configuration that minimises the local cost, given by the equation: $w_1 \times e_i + w_2 \times t_i$	Objective

the robot identifies the amount of energy (E_i) that can be used during a duty round to achieve system requirements. As such, the local contribution summary from robot i has the form

$$\langle (t_{i,1}, e_{i,1}), (t_{i,2}, e_{i,2}), \ldots, (t_{i,k_i}, e_{i,k_i}), (E_i) \rangle$$

Identifying the contribution summary entails that each robot uses probabilistic model checking to define the value for the attributes shown in Table 3.3. The table illustrates the PCTL formulas used to establish the values for the attributes used to form the contribution of robot i. To partition

Table 3.3: PCTL formulas for establishing contribution summary

Attribute	Description	PCTL formula
e_i	A *cost reward* for the total energy required by robot$_i$ to travel one meter of mission pipeline and provide a sensor reading	R"Energy"=? [F "done"]
t_i	A *cost reward* for the total time required by robot$_i$ to survey one meter of mission pipeline	R"Time"=? [F "done"]

system goals, linear programming method is used to model and solve the linear optimisation problem. After the partition of system goals, the i-th robot deals with varying probability of its sensor accuracy p_{il}^{retry}, by modifying its configurable speed of motion $sp_i \in (0, sp_i^{max}]$, adjusting its sensor configuration $p_i, 1 - p_i$. For example, a robot can alter speed to compensate for the degradation in the frequency of accurate sensor readings. Table 3.4 summarises parameters included in DTMC model that describes the behaviour of a robot. Some of the parameters are

configurable (i.e, those that the robot can change), while the reset of parameters are observable (i.e, a robot can not alter the value, but it can only observe the change in the value of the parameter).

Table 3.4: Parameters of robot$_i$ abstract DTMC model

Parameter	**Description**	**Type**
$speed$	A *variable* for the maximum speed a robot$_i$ can travel	Configurable
$distance$	A *constant* defined as a meter of the pipeline length and is used to calculate the time of travel(t_{travel})	Constant
p^{retry}	A *probability* of failure in robot$_i$ sensor	Observable

We assume that the $m = 3$ robots vary in terms of capabilities in performing each type of inspection task as follows:

1. Robot$_1$ is equipped with ($m_1 = 2$) sensors and only capable of executing inspection tasks of type $taskTyp_m$

2. Robot$_2$ is deployed with ($m_2 = 2$) sensors and only capable of executing inspection tasks of type $taskTyp_p$

3. Robot$_3$ is equipped with ($m_3 = 4$) sensors and capable of executing inspection tasks of type $taskTyp_m$ and $taskTyp_p$

Table 3.5: Capability summaries exchanged by robots

Robot	**Capability summary**
Robot$_1$	$\langle(e_0,t_0),(\mathsf{nil},\mathsf{nil}),(E_0)\rangle$
Robot$_2$	$\langle(\mathsf{nil},\mathsf{nil}),(e_1,t_1),(E_1)\rangle$
Robot$_3$	$\langle(e_2,t_2),(e_3,t_3),(E_2)\rangle$

Table 3.5 shows the capability summaries exchanged by the three robots. As shown in the table, each robot reports its potential contribution to fulfilling system requirements based on its sensors' ability to carry out different types of tasks. The goal partitioning problem is to allocate to robot i segments of length $x_{i,1} \leq L_1$ and $x_{i,2} \leq L_2$ for inspection associated with the two types of gas at the oil refinery, such that the system accomplishes its mission. Note that in this example of a distributed SAS, we have $x_{i,3} = 0$, since the last component of the capability summary is only used to provide information about the energy that each robot has.

The constraints (3.3) and optimisation objective (3.4) for this case study are as follows:

1. C is a set of $l_i = 8$ constraints that can be encoded in a linear program model. The model includes the decision variables shown in Table 3.6, which illustrates the problem variables that are used to formulate the constraints and objective function of the linear optimisation problem. In general, the problem described in previous section for partitioning system

requirements can be modeled and solved using LP with 8 constraints of the form from (3.3), with the functions F_j, $\bowtie_j$ and $bound_j$ defined as follows:

$$F_1(i,k,c_{i,k}) = \begin{cases} 1, & \text{if } k = 1 \wedge c_{i,k} \neq (\text{nil},\text{nil}) \\ 0, & \text{otherwise} \end{cases}, \text{ and } \; bound_1(\cdot) = L_1.$$

$$F_2(i,k,c_{i,k}) = \begin{cases} 1, & \text{if } k = 2 \wedge c_{i,k} \neq (\text{nil},\text{nil}) \\ 0, & \text{otherwise} \end{cases}, \text{ and } \; bound_2(\cdot) = L_2.$$

For $j \in \{3,4,5\}$, we have:

$$F_j(i,k,c_{i,k}) = \begin{cases} e, & \text{if } j = i+2 \wedge k \in \{1,2\} \wedge c_{i,k} = (e,t) \neq (\text{nil},\text{nil}) \\ 0, & \text{otherwise} \end{cases},$$

and, if $c_{j-2,3} = (E)$, then

$$bound_j\left(\left\{\langle c_{1,1},\ldots,c_{1,k_1},\ldots,c_{m,1},\ldots,c_{m,k_m}\rangle\right\}_{i=1}^{m}\right) = E.$$

Finally, $j \in \{6,7,8\}$, we have:

$$F_j(i,k,c_{i,k}) = \begin{cases} t, & \text{if } j = i+5 \wedge k \in \{1,2\} \wedge c_{i,k} = (e,t) \neq (\text{nil},\text{nil}) \\ 0, & \text{otherwise} \end{cases}, \text{ and } \; bound_j(\cdot) = T_{Round}.$$

These instantiations of the general format of a constraint (3.1) map to the following application-specific constraints:

$$C_1: \quad x_{1,1} + x_{3,1} \leq L_1,$$

i.e., robots 1 and 3 (which have methane sensors) need to jointly cover a distance $x_{1,1} + x_{3,1}$ not exceeding L_1. Likewise, for the second constraint we have:

$$C_2: \quad x_{2,2} + x_{3,2} \leq L_2,$$

which specifies that robots 2 and 3 need to jointly cover a distance $x_{2,2} + x_{3,2}$ not exceeding L_2. The next three constraints specify that none of the robots should be allocating work that requires more energy to carry out than the robots have available:

$$\begin{aligned} C_3: &\quad e_{1,1}x_{1,1} \leq E_1, \\ C_4: &\quad e_{2,2}x_{2,2} \leq E_2, \\ C_5: &\quad e_{3,1}x_{3,1} + e_{3,2}x_{3,2} \leq E_3, \end{aligned}$$

Finally, the last three constraints specify that the robots should cover the assigned distance and not exceeding time (T_{round}) allocated for a duty round:

$$\begin{aligned} C_6: &\quad t_{1,1}x_{1,1} \leq T_{round}, \\ C_7: &\quad t_{2,2}x_{2,2} \leq T_{round}, \\ C_8: &\quad t_{3,1}x_{3,1} + t_{3,2}x_{3,2} \leq T_{round}, \end{aligned}$$

2. The optimisation objective (i.e., maximising the distance covered by the three robots taken

together) using LP objective function of the form from (3.4), with the optimisation objective g defined as follows:

$$\text{maximise } g: \quad \sum_{i=1}^{3}\sum_{k=1}^{2} u_k x_{i,k}$$

where u_1 and u_2 are the utilities associated with covering one metre of methane and propane pipeline, respectively (see requirement R3).

Table 3.6 shows the decision variables $x_{i,k}$ that are non-zero according to the constraints above.

Table 3.6: Decision variables of linear optimisation problem

Variable ($x_{i,k}$)	**Description**
$x_{1,1}$	A variable encoding the total distance units assigned to robot$_1$ to execute inspection task of type $taskTyp_m$ ($k = 1$)
$x_{2,2}$	A variable encoding the total distance units assigned to robot$_2$ to execute inspection task of type $taskTyp_p$ ($k = 2$)
$x_{3,1}$	A variable encoding the total distance units assigned to robot$_3$ to execute inspection task of type $taskTyp_m$ ($k = 1$)
$x_{3,2}$	A variable encoding the total distance units assigned to robot$_3$ to execute inspection task of type $taskTyp_p$ ($k = 2$)

Finally, Table 3.7 describes the problem parameters and constraints introduced as part of formulating the partition problem using linear programming method. In this table, L_1, L_2, T_{round} are upper bound thresholds that are known by all robots prior the deployment to the mission.

The actual implementation and an extensive evaluation of the case study defined in this section are presented later in the thesis, in Chapter 5.

3.4 Goal partitioning using integer programming

3.4.1 Theoretical foundation

Integer programming (**IP**) is a mathematical technique dedicated for problems that include *discrete* decision variables that are used to formulate non-linear equations and inequalities. If a problem can be formulated as a mathematical equations of an integer program, the method is used to find the best solution to the problem. IP problems include a set of constraints that may take the form of non-linear inequality and objective function. This section illustrate the use of IP method to partition the goals of a system comprising distributed $m \geq 2$ components (e.g., teams of mobile robots, sensors in IoT system, etc.). Unlike LP method, IP method include discrete decision variables, while the variables in a Linear programming model are continuous.

For instance, in the scope of our work, the decision variables of an integer program problem model have to be discrete to represent actions such as to which robot a task should be assigned and

Table 3.7: Description of LP problem variables and constraints

Variable\ Constraint	**Description**	**Unit**
$x_{1,1}$	A *variable* for the total distance assigned to robot_1 from the pipeline perimeter that is used to extract methane (L_1)	metre
$x_{2,2}$	A *variable* for the total distance assigned to robot_2 from the pipeline perimeter that is used to extract propane (L_2)	metre
$x_{3,1}$	A *variable* for the total distance assigned to robot_3 from the pipeline perimeter that is used to extract methane gas (L_1)	metre
$x_{3,2}$	A *variable* for the total distance assigned to robot_3 from the pipeline perimeter that is used to extract propane gas (L_2)	metre
L_1	An *upper bound* for the total distance of mission methane pipeline	metre
L_2	An *upper bound* for the total distance of mission propane pipeline	metre
$e_{1,1}$	Energy cost incurred by robot_1 to travel and provide a sensor reading for 1 metre of methane pipeline	Joules/metre
$e_{2,2}$	Energy cost incurred by robot_2 to travel and provide a sensor reading for 1 metre of propane pipeline	Joules/metre
$e_{3,1}$	Energy cost incurred by robot_3 to travel and provide a sensor reading for 1 metre of methane pipeline	Joules/metre
$e_{3,2}$	Energy cost incurred by robot_3 to travel and provide a sensor reading for 1 metre of propane pipeline	Joules/metre
E_1	An *upper bound* for the total amount of energy available to robot_1 that can be utilized in a duty round	Joules/T^{round}
E_2	An *upper bound* for the total amount of energy available to robot_2 that can be utilized in a duty round	Joules/T^{round}
E_3	An *upper bound* for the total amount of energy available to robot_3 that can be utilized in a duty round	Joules/T^{round}
$t_{1,1}$	Time required by robot_1 to survey distance unit of methane pipeline	seconds/metre
$t_{2,2}$	Amount of time required by robot_2 to survey distance unit of propane pipeline	seconds/metre
$t_{3,1}$	Amount of time required by robot_3 to survey distance unit of methane pipeline	seconds/metre
$t_{3,2}$	Amount of time required by robot_3 to survey distance unit of propane pipeline	seconds/metre
T_{Round}	An *upper bound* for the total amount of time available to robots that can be utilized in a duty round	seconds
u_1	A *utility* that measures preference of monitoring methane pipeline	NP
u_2	A *utility* that measures preference of monitoring propane pipeline	NP

how many tasks should be assigned to each robot. Such problem model typically include binary variables that maps the assignment of tasks to robots. In particular, a binary decision variable that may takes the values 0 or 1, is used to represent the assignment of a given task to a robot. Also, the integer program model may include integer variables that represent the quantities of the assigned tasks to each robot. Whereas decision variables of a linear programming problem have to be continuous to capture decisions such as determining the length of distance assigned to each robot [56]. In the scope of our work, we consider an IP problem that includes variables that are restricted to be an integer. This problem typically is referred to as a *linear* integer-programming problem, which can be used to formulate and solve many real-world problems [24]. As with solving linear program problems, the process for solving integer program problems require finding the adequate formalism for the problem. Once an adequate formalism is arranged, a software solver can be used to obtain the answer to the problem.

To use integer programming method to partition system goals within the nuDECIDE approach, the constraints f_j and optimisation objective g from (3.1) and (3.2), respectively, must be of the appropriate form. Specifically, all the equations and inequalities used to formulate the constraints f_j and optimisation objective g should be of an integer form as follows.

$$f_j: \quad \sum_{i=1}^{m}\sum_{k=1}^{k_i} y_{i,k}F_j(i,k,c_{i,k}) \leq bound_j\left(\left\{\langle c_{1,1},\ldots,c_{1,k_1},\ldots,c_{m,1},\ldots,c_{m,k_m}\rangle\right\}_{i=1}^{m}\right), \tag{3.5}$$

where:

- $y_{i,k} \in \mathbb{Z}_{\geq 0}$ is a *discrete* variable used to quantify the decision of a problem;
- $F_j(i,k,c_{i,k}) \in \mathbb{R}_{\geq 0}$ is a function that takes the k-th contribution (2.14) of component i and returns a non-negative value based on the attribute values within this contribution;
- $bound_j\left(\left\{\langle c_{1,1},\ldots,c_{1,k_1},\ldots,c_{m,1},\ldots,c_{m,k_m}\rangle\right\}_{i=1}^{m}\right)$ is a function that returns a non-negative value based on the contribution summaries of all components.

$$\text{minimise/maximise } g: \quad \sum_{i=1}^{m}\sum_{k=1}^{k_i} c_{i,k}y_{i,k} \tag{3.6}$$

where $c_{i,k} \geq 0$ is a "cost" associated with component i and the k-th tuple from the contribution summary of component i. These costs are coming from the system-level requirements (i.e., the system-level optimisation objective).

The purpose of this section is to present a class of goal partitioning problems that can be represented and solved using integer programming formulations. In particular, we present a candidate assignment problem and illustrate how to formulate a goal partition problem that corresponds to this class of integer program problems. Typically, an assignment problem is described as combinatorial optimization problem [187] and the problem comprises several components that need to partition several tasks. An assignee is a productive unit that can be assigned a task or more for execution. The goal of the assignment problem is to find the optimal partition that

maximises or minimise a common objective. Typically in such problems, the components vary in terms of the qualitative metrics such as energy, time and utility associated with executing a task. Also, the assignment of a task in such problems is restricted to at most one assignee. The objective function in an assignment problem model encodes optimization of metrics like time or energy consumed, which are factored in as costs associated with the synthesis of a partition plan.

Moreover, the problem model includes a set of constraints that place constraints on the resources (e.g, energy, time, etc.) that an assignee can consume to perform task(s). In general, assignment problems are categorised into *balanced* and *unbalanced* problems [150]. An assignment problem is described as balanced when the number of available tasks is equal to the number of components and the assignment is carried out by assigning at most one assignee to each task. Also, each task can only be assigned to at most one assignee to be executed. for instance, a balanced assignment problem may comprise n components and n tasks. A feasible partition for such problem can be represented as a bipartite graph that includes n vertices, which link at most one component to one task and vice-versa. Whereas in an unbalanced assignment problem, a component may perform any number of tasks as the number of tasks is not equal to the number of components. Similar to a balanced problem, a task in unbalanced can only be assigned to at most one component.

There are various polynomial-time algorithms that are specialised to solve some categories of the assignment problems and obtain optimal partition that satisfy problem constrains [122, 139]. The simplex algorithm [109], which is a popular mathematical optimization technique for linear programming problems, can also be used to solve this assignment problems under certain conditions. These conditions require that the assignment problems can be modeled as a transportation problem, which in turn can be modeled and solved as a special case of a linear program. Solving an assignment problem typically entails checking all the possible combination of assignments and verify if a given candidate solution satisfy system constraints to be considered as a feasible solution. An objective function is used to rank all the feasible solutions and determine an optimal partition that maximise/minimise some criteria. In general, an integer program problem is considered to be NP-complete, in which tackling problems comprising relatively large number of variables and constrains using brute force technique is unfeasible.

Thus, the raise in the number of decision variables and constraints of an assignment problem increases the number of possible assignments that need to considered. Exploring all combination of assignments at run-time using brute-force technique is inefficient as the calculation may lead to computation overhead and delays in the partition of system goals. The situation exacerbates if the assignment problem includes a set of constraints that need to be satisfied to consider if each assignment is a feasible solution or not. To overcome this problem nuDECIDE employs computationally efficient mathematical techniques such as integer and linear program to formulate and solve partition problems. These techniques are applied in a variety of real-world problems such capital-budgeting [24], production line scheduling [147, 162], logistics and transportation prob-

lems [189] to solve NP-hard problems and obtain optimal or near-optimal solutions in polynomial time [125].

We demonstrate in the following Section 3.4.2, an example comprising an unbalanced assignment problem, which illustrates how nuDECIDE can support the use of integer program method to partition system requirements. The example demonstrates the characteristics of system requirements that can be formulated as an IP problem.

3.4.2 Example

In this section, we illustrate the application of the theoretical results from Section 3.4.1 to a system comprising $m = 3$ robots that are required to collect waste in a public park environment. This section provides the description of this system, and maps its associated goal partitioning problem to an integer programming problem Later in Chapter 5, we will present a complete implementation and evaluation of this case study, using three simulation based robots.

The park contains $n \geq 1$ physically distributed waste collections tasks and the m-robot system is equipped with the necessary tools to execute the n-th task. Each robot i, $1 \leq i \leq m$ has $m_i = 8$ modes of operation, which allows to associate a task execution with different time/energy use trade-offs. Each operation mode k, $1 \leq k \leq m_i$ of robot i requires $t_{i,k}$ time units and consumes energy $e_{i,k}$ to perform a task. The n-th task contains three waste collection sub-tasks [$subT_{1n}$, $subT_{2n}$, $subT_{3n}$]. The first sub-task ($subT_{1n}$) is mandatory to execute and requires the robot to collect general waste. Whereas the second and third sub-tasks ($subT_{2n}$ & $subT_{3n}$) are optional to perform and entails that the robot collects full and non-full bins of recyclable waste, respectively.

Figure 3.2 illustrates a CTMC model that describes the behaviour of robot i in carrying out the n-th task. In the initial state (s_0) of this model, the robot reaches the location of the n-th task with rate λ_{move} (modelled by a transition to state (s_1)). In state (s_1), the robot performs sub-task ($subT_{1n}$) with rate $\lambda_{subT_{1n}}$ (modelled by a transition to state (s_2)). In state (s_2), the robot may encounter sub-task ($subT_{2n}$) with probability p_{full} (modelled by a transition to state (s_3)) or sub-task ($subT_{3n}$) with probability 1 - p_{full} (modelled by a transition to state (s_4)).

In state (s_3), the robot may choose to perform sub-task $subT_{2n}$ (i.e., collect *full bin* of recyclable waste) with probability p_{1i} (modelled by a transition to state s_5), or otherwise avoid executing the sub-task with probability $1 - p_{1i}$ (modelled by a transition to state (s_7). Similarly, when the robot encounters sub-task $subT_{3n}$, it can determine whether to collect *non-full bin* of recyclable waste with probability p_{2i} (modelled by a transition to state (s_6)), or otherwise not collecting the waste with probability $1 - p_{2i}$ (modelled by a transition to state (s_7). The CTMC is augmented with two reward structures, an "energy use" reward structure, and a "time" reward structure. In state s_0 of this model, the reward structures are associated with the state to find the energy and time consumed by the robot to visits the location of the n-th task. Also, the "energy use" and "time" reward structures are associated with states s_1, s_5 and s_6 to determines the energy and

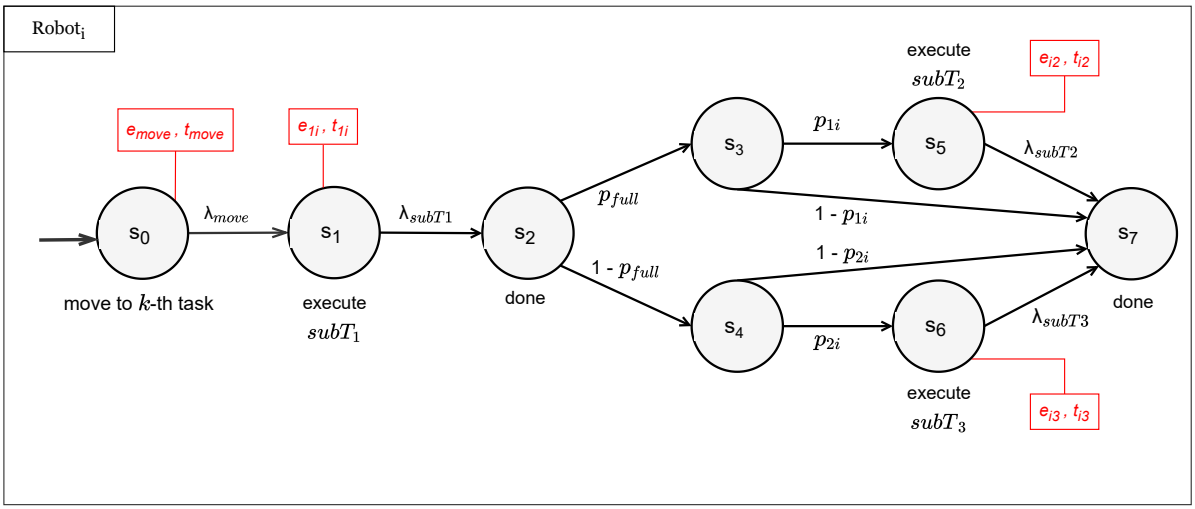

Figure 3.2: A CTMC model describing the behaviour of the i-th robot. Typically transitions in a CTMC model are associated with rates. However in states s_2, s_3, and s_4 of this model, rates are substituted with probability distributions to capture the branching of the events that a robot may encounter (as in state s_2 a robot may encounter a full or non-full bins (i.e., modelled by the transitions to state s_3 and s_4, respectively) and the decisions that the robot may make (as in states s_3 and s_4 a robot may choose to collect a full and non-full bins). Though in the actual CTMC model probabilities p_{full}, p_{i1} and p_{i2} are multiplied by a constant rate λ_c, whose value is large enough to avoid affecting time-related properties.

time consumed by the robot to execute sub-tasks $subT_{1n}$, $subT_{2n}$ and $subT_{3n}$.

We assume that the robot can configure parameters p_{1i} and p_{2i}, which define the transition probabilities in states s_3 and s_4, respectively. We further assume that robot i moves between the n-task locations with a configurable speed $sp_i \in (0, sp_i^{max}]$. The robot speed has an affects on the rate, in which the robot reaches the location of a task. In state (s_2), parameter p_{full} is considered an environment parameter, which determines the probability in which the robot may encounter a full or non-full recyclable waste bins (i.e., the likelihood in which the robot may be able to execute sub-tasks $subT_{2n}$ and $subT_{3n}$, respectively). A successful completion of the mission requires the collaboration among the team of robots to meet the system-level requirements from Table 3.9. Along with these system-level requirements,each robot i must satisfy its local requirements from Table 3.8.

The CTMC model shown in Figure 3.2 is used by robot i to identify alternative contributions that are characterised by the system constraints and objective shown in Table 3.9. In particular, robot i defines and shares its contribution summary per its modes of operation $1 \leq k \leq m_i$, which takes the form

$$\langle (t_{i,1}, e_{i,1}), (t_{i,2}, e_{i,2}), \ldots, (t_{i,k_i}, e_{i,k_i}), (E_i) \rangle$$

These contribution summaries have to satisfy robot's local constraints and identified using a local optimisation function shown in Table 3.8. The optimisation function selects a contribution

Table 3.8: Component-level requirements of multi-robot system deployed in a waste collection mission

Requirement	**Description**
$C1_i: \quad \forall 1 \leq i \leq m \,.\, \mathrm{b_i} \geq \frac{1}{4} B_i$	The robot should ensure that the energy left in its battery (b_i) is greater than or equal a quarter of its fully charged battery (B_i), so that to ensure a successful return of the robot to its charging dock
$O1_i: \forall\, 1 \leq k \leq m_i$, maximise $\sum_{k=1}^{m_i} w_1 p_{1,k_i} + w_2 p_{2,k_i} - w_3 e_{k_i} - w_4 t_{k_i}$	Subject to the constraints C_{1_i} being satisfied by multiple configurations, the i-th robot should use one of these configurations that maximises the utility function. where e_{k_i} and t_{k_i} are the total consumed energy and time associated with visiting and executing tasks using the k-th operating mode of robot i. $w_1, \ldots, w_4$ are weight constants to choose per each operation mode k, an optimal configuration between several alternatives

Table 3.9: System-level requirements of multi-robot system deployed in a waste collection mission

Requirement	**Description**
$C_2: \quad \forall\, 1 \leq i \leq m, \sum_{k=1}^{m_i} y_{i,k_i} e_{i,k_i} \leq E_i$	Each robot i must use energy less than or equal a threshold (E_i), to execute its tasks per its operating modes $y_{i,k_i} e_{i,k_i} + y_{i,k_i+1} e_{i,k_i+1} + \cdots + x_{i,m_i} e_{i,m_i}$
$C_3: \quad \forall\, 1 \leq i \leq m, \sum_{k=1}^{m_i} y_{i,k_i} t_{i,k_i} \leq T_{round}$	The m robots should execute their assigned tasks within the time available to complete a duty round T_{round}
$O_2: \quad$ miminise $\sum_{i=1}^{m} \sum_{k=1}^{m_i} y_{i,k_i} e_{i,k_i}$	Subject to the constraints C_{2_i} and C_{3_i} being satisfied by multiple partitions, the m-robot system should use one of these possible partitions that minimises the energy consumption associated with carrying out the tasks by the robots

per each operating mode of robot i. Those contribution summaries capture the timeliness and energy characteristics associated with executing a task. Table 3.10 shows the characteristics of the attributes in a contribution summary of robot i along with domain value of these attributes.

The task partitioning problem is to allocate the $n \geq 1$ tasks to y_{i,k_i}, $y_{i+1,k+1}$, $\ldots$, y_{m,m_i} robots 1 to m, per operating modes 1 to k_i, such that the system accomplishes its mission. Note that in this example of a distributed SAS, we have $y_{i,j_i+1} = 0$, since the last component of the capability summary is only used to provide information about the energy that each robot has. The constraints 3.5 and optimisation objective 3.6 for this case study are as follows:

1. C is a set of $l_i = 2$ constraints that can be encoded in a integer program model. Table 3.10 illustrates the form of the local contribution summary shared by robot i. As shown in the table, each robot has $m_i = 8$ operation modes. These operation modes are used to formulate the decision variables which are used to formulate the constraints and objective function of the integer program problem. In general, the problem described in previous section for partitioning system requirements can be modeled and solved using IP with 2 constraints of the form from (3.5), with the functions F_j, $\bowtie_j$ and $bound_j$ defined as follows:

   For $j \in \{1,2,3\}$, we have:

$$F_j(i,k,c_{i,k}) = \begin{cases} e, & \text{if } j = i \wedge 1 \leq k \leq 8 \wedge c_{i,k} = (e,t) \neq (\mathsf{nil},\mathsf{nil}) \\ 0, & \text{otherwise} \end{cases},$$

and, if $c_{j,9} = (E)$, then

$$bound_j\left(\{\langle c_{1,1},\ldots,c_{1,k_1},\ldots,c_{m,1},\ldots,c_{m,k_m}\rangle\}_{i=1}^{m}\right) = E.$$

Finally, $j \in \{4,5,6\}$, we have:

$$F_j(i,k,c_{i,k}) = \begin{cases} t, & \text{if } j = i+3 \wedge 1 \leq k \leq 8 \wedge c_{i,k} = (e,t) \neq (\mathsf{nil},\mathsf{nil}) \\ 0, & \text{otherwise} \end{cases}, \text{ and } bound_j(\cdot) = T_{Round}.$$

These instantiations of the general format of a constraint 3.5 map to the following application-specific constraints:

$$C_2: \qquad \forall\ 1 \leq i \leq m, \sum_{k=1}^{m_i} y_{i,k_i} e_{i,k_i} \leq E_i$$

This constraint specifies that the m robots should perform their assigned tasks per operating modes $y_{i,k}e_{i,k} + y_{i,k+1}e_{i,k+1} + \cdots + y_{i,m_i}e_{i,m_i}$ should be less than or equal an upper bound (E_i). Finally, the last constraint specifies that the robots should complete their assigned tasks not exceeding time (T_{round}) allocated for a duty round:

$$C_3: \qquad \forall\ 1 \leq i \leq m, \sum_{k=1}^{m_i} y_{i,k_i} t_{i,k_i} \leq T_{round}$$

2. The optimisation objective (i.e., minimising the energy consumed by the m robots) using IP objective function of the form 3.6, with the optimisation objective g defined as follows:

$$O: \qquad \text{minimise } g: \sum_{i=1}^{m}\sum_{k=1}^{m_i} y_{i,k_i} e_{i,k_i}$$

Finally, the actual implementation and an extensive evaluation of the case study defined in this section are presented later in the thesis, in Chapter 5.

Table 3.10: Capability summaries exchanged by the m-robot system

Robot$_i$	**Capability summary**
Robot$_1$	$\langle(e_{1,1},t_{1,1}),(e_{1,2},t_{1,2}),\ldots,(e_{1,8},t_{1,8}),(E_1)\rangle$
Robot$_2$	$\langle(e_{2,1},t_{2,1}),(e_{2,2},t_{2,2}),\ldots,(e_{2,8},t_{2,8}),(E_2)\rangle$
Robot$_3$	$\langle(e_{3,1},t_{3,1}),(e_{3,2},t_{3,2}),\ldots,(e_{3,8},t_{3,8}),(E_3)\rangle$

Execution of Local Control

After a successful allocation of mission tasks using the previously described method, the method determines the contribution of each component of the system to achieve system goals. In particular, the execution of the method produces a contribution level agreement (CLA), which defines the tasks that has been assigned to each robot along with the timeliness and energy efficiency characteristics that each robot must adhere to when performing its tasks. The local control instance running on each robot updates the system goals based on the contribution level agreement

that shows the contribution of each of the m system components to satisfy the system constraints and objectives. Throughout the mission, the m-robot system should satisfy mission constraints and objectives at all times. In such a dynamic environment, each robot i should adapt to changes in the probability of encountering more full recyclable bins (p_{full}), by continually adjusting:

1. The robot speed, which can be configured to take any of the values $sp_i \in (0, sp_i^{max}]$.

2. The probability (p_{1i}) of executing sub-task $subT_2$.

3. The probability (p_{2i}) of performing sub-task $subT_3$.

3.5 Goal partitioning using MDP policy synthesis

3.5.1 Theoretical foundation

Many of today's systems include distributed components that cooperate to achieve system goals and handle changes that occur in environments that are characterised by high levels of uncertainty. For instance, robots may encounter changes, such as the emergence of obstacles, a sudden decrease in the amount of energy available to complete tasks, or the failure of one of the robots that render it unable to perform any tasks. Therefore, partitioning goals among components in such systems requires the synthesis of an adaptation plan that takes into account the uncertainty (e.g., about the outcome of the execution of tasks, timeliness, etc). For instance, a robot might be equipped with a wheelbase that would not enable it to reach the location of task execution when weather conditions change. Overcoming such problems requires employing models capable of representing the problem of partitioning system goals and account for the possibility that system components may fail to satisfy their contribution to completing system goals. An additional challenge in this setting is providing assurances about the partition of goals, evidencing that components are indeed capable of satisfying system goals despite the inherent uncertainty that pervades the environment.

Given the aforementioned challenges, a suitable approach to partition system goals has to: (i) include a modelling formalism able to represent the allocation of system goals, (ii) compare and evaluate, based on some criteria (e.g., energy consumption, utility), the possible combinations of system goal partitions and select one that optimises some objective criteria, (iii) consider the (partially probabilistic) component behaviour required to satisfy their contribution to system goals, and (iv) provide formal guarantees that the allocation meets its functional and non-functional (e.g., timeliness, energy efficiency) constraints and objectives.

To satisfy these requirements, the goal partitioning approach proposed in this section employs MDP policy synthesis to allocate system goals to components. This is feasible when an MDP exists such that:

1. the alternative values for the contribution amounts $\{x_{i,1}, x_{i,2}, \ldots, x_{i,k_i}\}_{i=1}^{m}$ from the constraints (3.1) and optimisation objective (3.2) can be encoded as the parameters of an MDP;

2. the constraints (3.1) can be specified as PCTL properties $P_{\triangleright\triangleleft_j bound_j}[\cdot]$ or $R_{\triangleright\triangleleft_j bound_j}[\cdot]$ over the MDP;

3. the optimisation objective (3.2) requires the maximisation or minimisation of a PCTL-encoded quantity $P_{=?}[\cdot]$ or $R_{=?}[\cdot]$ over the MDP.

If these conditions are met, Figure 3.3 illustrates the steps involved in synthesizing the policy for allocating system goals, which aims to partition n tasks by assigning each task to one of the m system components, so that system functional and non-functional goals are satisfied. In particular, the use of the MDP policy synthesis approach entails:

(i) Producing an MDP model that encodes the possible assignments of the n tasks to the $\sum_{i=1}^{m} k_i$ component contributions/modes. The alternative assignments are captured as a nondeterministic choice that leaves the task-contribution matching underspecified, and costs of assignments (e.g., time, energy consumption) are encoded as reward structures populated with data from component contributions.

(ii) Producing a probabilistic temporal logic (typically a numerical multi-objective query in PCTL) that captures the set of quantitative constraints that need to be satisfied to consider a feasible allocation of system goals, along with an optimisation objective that maximises a reward (e.g., or minimises cost) criteria.

(iii) Synthesising a goal partition policy using probabilistic model checking [118]. This step takes an input the MDP model and the probabilistic temporal logic specifications generated in the previous steps. The aim is to identify a policy, among all existing policies of the MDP, that satisfies all the quantitative constraints and optimises a quantitative objective such as utility, time, or energy consumed.

(iv) Translating the generated goal partition policy generated by the model checker into a plan that can be executed by the system. The policy is encoded as a DTMC that lists sequentially each task and the component/mode assigned to it. This translation process entails extracting the sequence of task/component-mode pairs that each component has to execute to contribute to satisfy system goals. It is worth noting that in some cases, the model checker will produce a probabilistic choice for the allocation of some tasks. In such cases, the translation process will involve an additional step, which is described later in Section 3.5.1.3.

In the following sections, we describe in detail the different steps listed above.

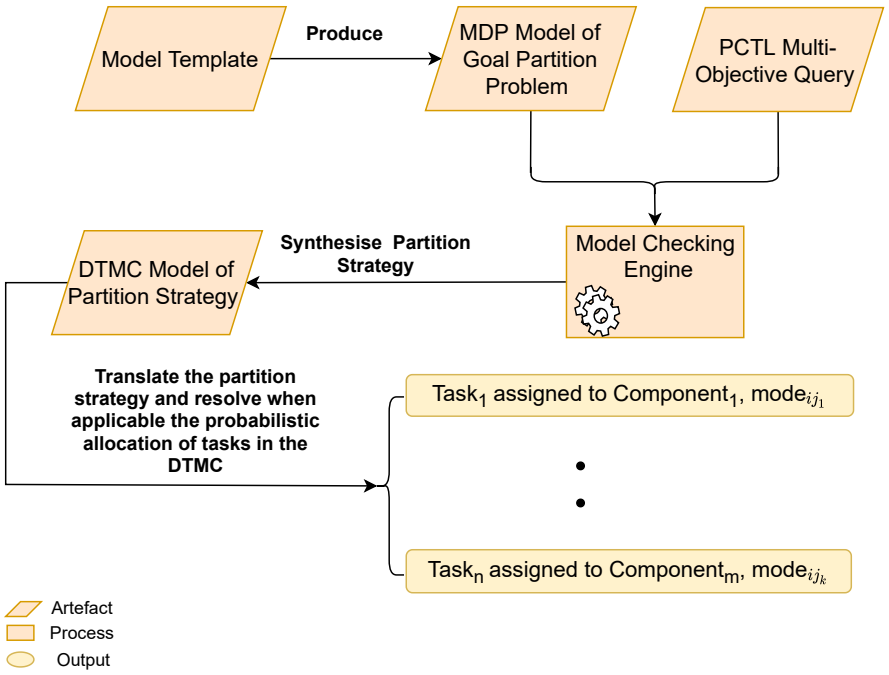

Figure 3.3: Synthesise Goal Partition Strategy

3.5.1.1 Goal Partition MDP Model Generation

To synthesise a goal partition policy, we define a high-level model of Markov decision processes that will be employed by our approach. This SMDP model is conveniently defined to closely resemble the high-level modelling constructs that are typically employed to describe MDP models in widespread probabilistic model checkers like PRISM [119] and Storm [65].

Definition 3.1. A Symbolic Markov Decision Process (SMDP) is a tuple $M = \langle V, \vec{v_0}, \Sigma, \delta, \iota \rangle$ where $V = \langle v_1, \ldots, v_n \rangle$ is a tuple of variables, $\vec{v_0} \in \mathscr{D}_V$ gives the initialisation conditions on the variables, Σ is a finite, non-empty set of actions, and δ is a finite set of probabilistic transitions $\langle \phi, \sigma, \mathscr{P}_U \rangle$ where $\phi \subseteq \mathscr{D}_V$ is a predicate on V which guards the transition, $\sigma \in \Sigma$ is the transition action, and $\mathscr{P}_U$ is a discrete probability distribution function over the set of update functions U, which are defined as a set of assignments of the form $u : \mathscr{D}_V \to \mathscr{D}_V$. Finally, $\iota : ID \times \mathscr{D}_V \times \Sigma \to \mathbb{R}_{\geq 0}$ is a reward structure that can be used to model different rewards or costs identified by $id \in ID$, in which transition actions in Σ fired from states that satisfy a predicate built on V accrue a non-negative reward/cost.

Based on that definition, we can automatically build a SMDP model by mapping elements from a problem instance like the one defined in Section 3.5.2, into structural elements of the SMDP.

Algorithm 3.5.1.1 generates a SMDP partition model that can be employed to synthesise a partition policy using a probabilistic model checker.

Algorithm 1 SMDP partition goal model generation

1: $V := \{t_{alloc}\} \cup \{s_{r_1}, \ldots, s_{r_m}\}$
2: $\overrightarrow{v_0} := \{(t_{alloc}, 0), (s_{r_1}, idle), \ldots, (s_{r_m}, idle)\}$
3: $\Sigma := \emptyset$
4: $\delta := \{(t_{alloc} < n \wedge allidle, \epsilon, \{(1, t_{alloc} := t_{alloc} + 1)\})\}$
5: $\iota := \emptyset$
6: **for all** $i \in \{1 \ldots m\}$ **do**
7: $\quad \delta := \delta \cup \{(t_{alloc} < n \wedge allidle, \epsilon, \{(1, \{t_{alloc} := t_{alloc} + 1, s_{r_i} := try\})\})\}$
8: $\quad \delta := \delta \cup \{(engaged(i, try), \epsilon, \{(p_{cij}(c_{i,j}, t_{alloc}), \{s_{r_i} := assign\}), (1 - p_{cij}(c_{i,j}, t_{alloc}), \{s_{r_i} := fail\})\})\}$
9: $\quad \delta := \delta \cup \{(engaged(i, fail), \epsilon, \{(1, \{s_{r_i} := idle\})\})\}$
10: $\quad$ **for all** $j \in \{1 \ldots k_i\}$ **do**
11: $\quad\quad \Sigma := \Sigma \cup \{c_{i,j}\}$
12: $\quad\quad \delta := \delta \cup \{(engaged(i, assign), c_{i,j}, \{(1, \{s_{r_i} := idle\})\})\}$
13: $\quad\quad$ **for all** $l \in RC$ **do**
14: $\quad\quad\quad \iota := \iota \cup \{(l, \top, c_{i,j}, f_{cij}(c_{i,j}, l))\}$
15: $\quad\quad$ **end for**
16: $\quad$ **end for**
17: **end for**
18: **return** $\langle V, \overrightarrow{v_0}, \Sigma, \delta, \iota \rangle$

The algorithm receives as input the set of component contributions $c_{i,j}, i \in \{1, \ldots, m\}, j \in \{i, \ldots, k_i\}$ (which will be referred to as CC in the remainder for simplicity), a function $p_{cij} : CC \times \mathbb{N} \rightarrow [0, 1]$ that returns the probability of successful completion of a task by component/robot mode, a set of labels RC that identify the different dimensions of concern (e.g., energy, time), and a function $f_{cij} : CC \times RC \rightarrow \mathbb{R}_{\geq 0}$ that returns a cost/reward associated with a given dimension of concern.

The first part of the algorithm deals with the initialisation of the different elements of the SMDP. In line 1, the set of variables is built as the union of variable t_{alloc} (which is used to keep track of the number of tasks already allocated and takes values in the domain $\{1, \ldots, n\}$), and a set of state variables for component/robots $s_{ri}, i \in \{1, \ldots, m\}$, which take values in the domain $\{idle, try, fail, assign\}$. In line 2, the initial values for the set of variables are set as 0 assigned tasks for t_{alloc}, and $idle$ for all system component state variables. In line 4, the set of probabilistic transitions δ is initialised with a transition that enables skipping the assignment of tasks to a system component by allowing to increment t_{alloc} if its value is smaller than the number of tasks n and none of the system's components are engaged in an assignment of tasks (encoded by $allidle \equiv \wedge_{i=1}^{m} s_{r_i} = idle$). Note that in this transition (and others in the algorithm), the action is labeled with the empty string denoted by ϵ, because the action associated with the transition is not relevant for the extraction of the partition policy. The set of actions Σ and reward structure ι are initially empty (lines 3 and 5). Note that the update of the variable in this and other transitions that are not probabilistic occurs with probability 1.

After initialisation, the body of the algorithm builds up both the set of probabilistic transitions δ, and the reward structure ι. Lines 7-9 add probabilistic transitions to δ that:

- (line 7) engage a system component r_i by changing its state to try if all components are idle and task assignments have not been completed.

- (line 8) if a component is engaged in state try (captured by function $engaged(x,s) \equiv \wedge_{i=1,i\neq x}^{m} s_{r_i} = idle \wedge s_{r_x}$), it changes its state to $assign$ with probability $p_{cij}(c_{i,j}, t_{alloc})$, which corresponds to the estimated probability of successfully completing task t_{alloc}, and alternatively, updates state to $fail$ with the complementary probability,

- (line 9) if task execution has failed, it returns the state of the component to $idle$.

In the inner loop of the algorithm (lines 10-16), we build:

- (line 11) the set of actions Σ that correspond to the different modes of system components,

- (line 13) the set of transitions that correspond to the selection of alternative modes $c_{i,j}$ for every component, and that return the state of the component to $idle$, and

- (lines 13-15) the reward structure ι that adds to every label l in RC that identifies a dimension of concern, a positive reward given by $f_{cij}(c_{i,j}, l)$, which is accrued whenever the action that corresponds to $c_{i,j}$ is fired from any state ($\top$ denotes that the guard is always satisfied).

The algorithm finishes by returning the SMDP tuple assembled with the elements built. It is worth noticing that this algorithm is also valid to solve a non-probabilistic version of the problem in which all probabilities in P are equal to one.

3.5.1.2 Probabilistic Temporal Logic Specification Generation

Having derived the MDP model for the decentralised SAS under development, each constraint and the optimisation objective for the system need to be encoded into PCTL. The result of this encoding is a multi-objective probabilistic logic query expressed in the extended PCTL specification language of the PRISM probabilistic model checker. The process is straightforward and is illustrated in (3.7).

3.5.1.3 Partition Policy Synthesis and Translation

The partition policy takes the form of a DTMC model that encodes the assignment of tasks to robot/component capabilities. Such policies may include probabilistic assignments as part of the task executions that correspond to a given component. Our approach turns the probabilistic assignment decision into a deterministic one according to the procedure described in Algorithm 3.5.1.3.

The algorithm receives as input a partition policy, which is a DTMC (S, s_i, P, AP, L) encoded as a set of tuples D that capture probabilistic transitions. Each tuple is of the form (s_s, s_t, p, l), where $s_s \in S$ is a source state, $s_t \in S$ is a target state, $p \in [0,1]$ is a probability, and l is an action label. As output, the algorithm produces a DTMC D' in the same format, in which all transitions are deterministic (i.e., with probability one).

The algorithm starts by adding to D' all tuples that correspond to deterministic transitions (line 4). If a transition that corresponds to a given tuple d is not deterministic, then the algorithm selects one of the possible outcomes of the transition by performing the following steps:

1. Function $f_{siblings}$ computes the set of transition tuples having the same source state as d, i.e., $f_{siblings}(s) \equiv \{d \in \mathrm{D} \mid d.s_t = s\}$ (line 6).

2. A set of probability intervals, each one associated to an action label extracted from the corresponding tuple, is computed for the set of sibling transitions (line 10).

3. One of the sibling transition tuples is selected and added to D' based on the selection function $select(x, intervals) \equiv \{i \in intervals \mid lo(i) < x \leq hi(i)\}$, where $lo(x)$ and $hi(x)$ return the low and the high bound of a probability interval (line 13). The generation of the argument $x\ in[0,1]$ provided to $select$ is done by generating a sequence of random numbers that follow a continuous uniform distribution [43]. Concretely, we define a function $generate_R : \mathbb{N} \rightarrow \mathbb{R}^n$ that produces a sequence of random fractions $\langle r_1, r_2, \ldots, r_n \rangle$, where $r \in \mathbb{R}_{\geq 0} : 0 \leq r \leq 1$. This function uses the Linear Congruential Method [110] to generate the uniformly distributed sequence of random fractions. Function get_R extracts one of such random values from the sequence.

The partition policy takes the form of a DTMC model that encodes the assignment of tasks to robot/component capabilities. Such policies may include probabilistic assignments as part of the task executions that correspond to a given component. Our approach turns the probabilistic assignment decision into a deterministic one according to the procedure described in Algorithm 3.5.1.3.

The algorithm receives as input a partition policy, which is a DTMC (S, s_i, P, AP, L) encoded as a set of tuples D that capture probabilistic transitions. Each tuple is of the form (s_s, s_t, p, l), where $s_s \in S$ is a source state, $s_t \in S$ is a target state, $p \in [0,1]$ is a probability, and l is an action label. As output, the algorithm produces a DTMC D' in the same format, in which all transitions are deterministic (i.e., with probability one).

Algorithm 2 Partition policy determinization

```
    D' := ∅
 2: for all d = (s_s, s_t, p, l) ∈ D do
      if d.p = 1 then
 4:     D' := D' ∪ {d}
      else
 6:     siblings := f_siblings(d.s_s)
        intervals := ∅
 8:     p_accrued := 0
        for all sibling ∈ siblings do
10:       intervals := intervals ∪ {(siblings.l, [p_accrued, p_accrued + siblings.p)}
          p_accrued := p_accrued + siblings.p
12:     end for
        D' := D ∪ {select(get_R(·), intervals)}
14:   end if
    end for
16: return D'
```

The algorithm starts by adding to D' all tuples that correspond to deterministic transitions (line 4). If a transition that corresponds to a given tuple d is not deterministic, then the algorithm selects one of the possible outcomes of the transition by performing the following steps:

1. Function $f_{siblings}$ computes the set of transition tuples having the same source state as d, i.e., $f_{siblings}(s) \equiv \{d \in \mathrm{D} \mid d.s_t = s\}$ (line 6).

2. A set of probability intervals, each one associated to an action label extracted from the corresponding tuple, is computed for the set of sibling transitions (line 10).

3. One of the sibling transition tuples is selected and added to D' based on the selection function $select(x, intervals) \equiv \{i \in intervals \mid lo(i) < x \leq hi(i)\}$, where $lo(x)$ and $hi(x)$ return the low and the high bound of a probability interval (line 13). The generation of the argument $x\ in[0,1]$ provided to $select$ is done by generating a sequence of random numbers that follow a continuous uniform distribution [43]. Concretely, we define a function $generate_R : \mathbb{N} \rightarrow \mathbb{R}^n$ that produces a sequence of random fractions $\langle r_1, r_2, \ldots, r_n \rangle$, where $r \in \mathbb{R}_{\geq 0} : 0 \leq r \leq 1$. This function uses the Linear Congruential Method [110] to generate the uniformly distributed sequence of random fractions. Function get_R extracts one of such random values from the sequence.

3.5.2 Example

To illustrate the use of MDP policy synthesis approach to partition system goals, we use the problem described in Section 3.4.2, which consists of an multi-robot system deployed for waste collection in a public park environment. For brevity, we only explain mission characteristics that deal with the partition of system goals and leave out aspects that relate to the individual behaviour of these robots, with the exception of the local constraints that each robot needs to satisfy, which are presented in Table 3.11. In addition to that, the table illustrates the utility

function used to identify the optimal contribution per operation mode $j_1, \ldots, j_{m_i}$ for robot i, as well as the parameters that influence its behaviour and ability to satisfy system goals.

Table 3.11: Characteristics of the multi-robot system

Characteristics of the mission and its requirements
The mission comprises N locations and each location includes two types of waste collection tasks $recyclable, general$
The task of collecting $general$ waste is mandatory to be executed, while the task of collecting $recyclable$ waste is optional to perform
The m-robot system must complete the tasks in all park environment regions within t minutes
Throughout the mission, robot $i, 1 \leq i \leq m$ checks if the energy left in its battery satisfies the constraint: $C_{1_i}: \quad \forall 1 \leq i \leq n \,.\, \mathrm{b}_i \geq \frac{1}{4} B_i$, where (b_i) is the energy left in the i-th robot battery and (B_i) is the energy available in its fully charged battery
If the previous constraint is satisfied by multiple configurations, the i-th robot should use one of these configurations that maximises the local utility, given by the utility function: $w_1 * p_{fullBin_i} + w_2 * p_{nonFullBin_i} - w_3 * e_{ti} - w_4 * t_{ti}$, where e_{ti} and t_{ti} are the total consumed energy and time associated with visiting and executing the tasks in a given location. $w_1, \ldots, w_4$ are weight constants to choose per each operation mode j, an optimal configuration between several alternatives
Characteristics of the i-th robot behaviour
The i-th robot moves with speed $sp_i \in (0, sp_i^{max}]$ between the N task locations and executes the j-th collection task $j \in recyclable, general$ with rate $r_{ij} > 0$
e_{ij}, t_{ij} refers to the energy and time consumed by robot i to execute j-th task
$p_{fullBin_i}$ is a robot configurable parameter that represents the probability of collecting *full recyclable* waste bin
$p_{nonFullBin_i}$ is a robot configurable parameter that represents the probability of collecting *non-full recyclable* waste bin

3.5.2.1 Task Allocation Problem Model

Listing 3.1, illustrates an instantiation of an MDP partitioning model for our example that results from applying Algorithm 3.5.1.1 using PRISM's concrete syntax. The model starts by defining the number of task assignments to be made (line 2) and the possible state values for robots, which correspond to those defined for Algorithm 3.5.1.1 $\{idle, try, assign, fail\}$ (line 3). Note that in the listing, parts highlighted in blue correspond to information provided as input to the Algorithm.

The model consists of a single process (*module* in PRISM terms) called `TaskAllocation`, which contains the variables that keep track of the tasks already allocated (`t_alloc`) and state of the different robots (`sr1-sr3`), as well as their initialization (lines 6-9). This part of the model is generated by lines 1-2 in Algorithm 3.5.1.1.

The rest of the process consists of a set of commands that encode how the assignment is carried out. In particular, the number of the task to which a robot mode is being assigned is kept in `t_alloc`, which is incremented every time a robot is engaged for task assignment (lines 11, 19), or whenever the task assignment is skipped (line 21). Note that there are no explicit task identifiers in the model, because the task-robot mode matching is implicit in the order of the actions in which actions are found in the MDP policy produced (e.g., if the first action found in the policy is `[r3c2]`, it means that task 1 is assigned to mode 2 of robot 3).

Commands in lines 11-14 correspond to the instantiation of transitions generated by Algorithm 3.5.1.1 in lines 7-9, and take care of engaging the component for assignment, and encoding the probabilistic action for successful execution of the task, respectively.

Lines 16-17 capture the possible assignments to a robot mode, which are encoded as a non-deterministic choice (generated by the algorithm in line 12).

Finally, lines 28-48 capture the different reward structures, which correspond to RC = {`task-utility, time-r1, ..., time-r3, energy-r1, ...,energy-r3`}.

```
1  mdp
2  const N = 8; // n tasks to assign
3  const IDLE=0; const TRY=1; const ASSIGN=2; const FAIL=3; // component/robot states
4
5  module TaskAllocation
6  t_alloc: [0..N] init 0; // keeps track of number of tasks assigned
7  sr1 : [IDLE..FAIL] init IDLE; // robot states
8  ...
9  sr3 : [IDLE..FAIL] init IDLE;
10 ...
11 [ ] (t_alloc<N) & (sr1=IDLE) & (sr2=IDLE) &(sr3=IDLE) -> (t_alloc'=t_alloc+1) & (sr1'=TRY); //
       engage r1
12
13 [ ] (sr1=TRY) & (sr2=IDLE) & (sr3=IDLE)    -> p_1:(sr1'=ASSIGN) + (1-p_1):(sr1'=FAIL); // r1 tries
       task
14 [ ] (sr1=FAIL) & (sr2=IDLE) & (sr3=IDLE) -> (sr1'=IDLE); // if r1 failed, returns to IDLE
15
16 [r1c1] (sr1=ASSIGN) & (sr2=IDLE) & (sr3=IDLE)    -> (sr1'=IDLE); // mode selection for task
17 [r1c2] (sr1=ASSIGN- & (sr2=IDLE) & (sr3=IDLE)    -> (sr1'=IDLE);
18 ...
19 [ ] (t_alloc<N) & (sr3=IDLE) & (sr1=IDLE) & (sr2=IDLE) -> (t_alloc'=t_alloc+1) & (sr3'=TRY); //
       engage r3
20 ...
21 [ ]  (t_alloc<N) & (sr1=IDLE) & (sr2=IDLE) & (sr3=IDLE) -> (t_alloc'=t_alloc+1); // skip assignment
       for task
22 endmodule
23
24 formula p_1 = t_alloc=0? (0.8 : (t_alloc=1?: 0.6: (...) );  // Probability of r1 succeeding at task
25 ...
26 formula p_3 = t_alloc=0? (0.5 : (t_alloc=1?: 0.7: (...) );  // Probability of r3 succeeding at task
27
28 rewards "task-utility" // reward structure for task utility
29 [r1c1] true: 0.63;
30 [r1c2] true: 0.86;
31 ...
32 [r2c1] true: 0.64;
33 ...
34 endrewards
35
36 rewards "time-r1" // reward structure for r1 timeliness
37 [r1c1] true: 16.64;
```

```
38 ...
39 endrewards
40 rewards "time-r3" // reward structure for r3 timeliness
41 [r2c1] true: 23.80;
42 ...
43 endrewards
44
45 rewards "energy-r1" // reward structure for r1 energy cost
46 [r1c1] true: 423.98;
47 ...
48 endrewards
49 ...
```

Listing 3.1: MDP model of task partition problem

3.5.2.2 Synthesis of Partition Policy

The synthesis of the partition policy requires the use of an MDP task allocation model like the one described in the previous section. Providing that model as input to a probabilistic model checker, we can synthesize a partitioning policy that divides the n tasks among the robots. The policy indicates the set of tasks assigned to each robot along with the operating mode associated with the execution of every task. The synthesis of the partition policy entails model checking the partition model against the multi-objective PCTL property shown in 3.7, which includes two parts: (i) A reward maximisation formula that indicates that the synthesis of partition policy should resolve the non-determinism in the model in such a way that it maximises the cumulative value of reward structure "*task-utility*" (lines 28-34). The structure captures quantifiable utility rewards which indicate that mission robots have collected more recyclable waste. (ii) A set of cumulative reward constraints that capture the limitation of the resources available (i.e., time, energy) for $r_1, \ldots, r_m$ robots to carry out their assigned tasks. These constraints are formulated per robot and takes the form shown in Table 3.12.

$$\text{multi}(\underbrace{R\{\text{"utility"}\}_{max=?}[C]}_{\text{Utility reward maximisation}}, \underbrace{R\{\text{"time"}\}_{r_1} \leq t\ [C],\ R\{\text{"energy"}\}_{r_1} \leq E_{r_1}\ [C], \ldots}_{\text{Reward constraints of the } i\text{−th robot}}) \tag{3.7}$$

Table 3.12: PCTL formulas for timeliness and energy efficiency concerns

Name	PCTL Formula	Description
ϕ_{t_i}	$R\{\text{"time"}\} \leq t\ [C]$	A constraint that checks, using a "*time*" reward structure that is robot specific, whether the time accumulated in minutes, as a result of the i-th robot executes its assigned tasks is less than an upper bound t.
ϕ_{e_i}	$R\{\text{"energy"}\} \leq E_i\ [C]$	A constraint that checks, using an "*energy*" reward structure that is robot specific, whether the energy consumed in joules, as a result of the i-th robot executes its assigned tasks is less than a robot specific upper bound e_i

In particular, the constraints are used to check if the cumulative value of some reward structures that capture timeliness and energy efficiency concerns are within bounds. The bounds that address timeliness concerns are specified as part of the mission characteristics in Table 3.11. While the bounds that correspond to the energy efficiency concerns are robot specific and defined based on the summary of contributions received from each robot.

3.5.2.3 Partition Policy Translation

Listing 3.2 shows an example of a partition policy DTMC encoded as a set of tuples. Notice that lines that do not include labels are assumed to be tagged with the empty string. In this DTMC, there is one probabilistic choice that can be observed in the tuples on lines 11-12, both corresponding to transitions that start in state 22.

```
1  0 4 1           // robot2WasAllocatedTask
2  1 9 1           // robot3WasAllocatedTask
3  4 5 1           // robot2Try
4  5 1 1 r2c8
5  8 16 1          // robot3WasAllocatedTask
6  9 10 1          // robot3Try
7  10 8 1 r3c8
8  15 23 1         // robot3WasAllocatedTask
9  16 17 1         // robot3Try
10 17 15 1 r3c8
11 22 30 0.106311  // robot3WasAllocatedTask
12 22 32 0.893689  // robot2WasAllocatedTask
13 23 24 1          // robot3Try
14 24 22 1 r3c8
15 29 39 1          // robot2WasAllocatedTask
16 30 31 1          // robot3Try
17 31 29 1 r3c8
18 32 33 1          // robot2Try
19 33 29 1 r2c8
20 36 46 1          // robot2WasAllocatedTask
21 39 40 1          // robot2Try
22 40 36 1 r2c8
```

Listing 3.2: DTMC partition policy example

To obtain a deterministic version of this policy, Algorithm 3.5.1.3 detects that these two tuples start from the same state and end up in different states with different probabilities (function $f_siblings$). At that point, the algorithm builds a set of intervals, that in this case would correspond to: $[0, 0.106311]$,$[0.106311, 1.0]$, and select a single transition based on the pseudo-random fraction returned from function get_R. Assuming that the value returned is 0.5, the resulting policy can be observed in Listing 3.3.

```
1  0 4 1           // robot2WasAllocatedTask
2  1 9 1           // robot3WasAllocatedTask
3  4 5 1           // robot2Try
4  5 1 1 r2c8
5  8 16 1          // robot3WasAllocatedTask
6  9 10 1          // robot3Try
7  10 8 1 r3c8
8  15 23 1         // robot3WasAllocatedTask
9  16 17 1         // robot3Try
10 17 15 1 r3c8
```

```
11 22 32 1          // robot2WasAllocatedTask
12  23 24 1          // robot3Try
13  24 22 1 r3c8
14  29 39 1          // robot2WasAllocatedTask
15  30 31 1 // robot3Try
16 31 29 1 r3c8
17  32 33 1          // robot2Try
18  33 29 1 r2c8
19  36 46 1          // robot2WasAllocatedTask
20  39 40 1          // robot2Try
21  40 36 1 r2c8
```

Listing 3.3: Deterministic partition policy example

In the deterministic version of the policy, it can be observed how the alternative transition from state 22 assigning the task to `r3` has disappeared and instead, a transition with probability 1 assigns the task to `r2` (highlighted in red, line 11). Unreachable transition tuples are grayed out (lines 15-16).

3.6 Selection of goal partitioning method

This section examines the characteristics of the introduced methods for solving various classes of the goal partitioning problem. These characteristics along with the partition problem examples introduced in sections 3.2.2, 3.3.2, 3.4.2, and 3.5.2 provide insight into the scope of applications that can be supported by each partition method. Those characteristics clarify the criteria on which the user should base his decision when choosing a method for solving an allocation problem.

The first method, exhaustive search, involves examining every possible partition of the system goals among its components and is applicable when only a finite and small number of such partitions are feasible. In particular, this method is tailored towards combinatorial optimisation and enumeration problems where the size of the decision variables and constraints is relatively small. Nievergelt [140] argues that the use of exhaustive search is often a necessity in handling these optimisation problems. As the state space of such optimisation problems often lacks any regular structures. The use of the exhaustive search method is straightforward and effective in solving optimisation problems comprising relatively small state space.

The use of this method complements the shortcoming found in the linear program, integer program and MDP policy synthesis methods. Although, formalising and solving a partition problem using the earlier method is conceptually simple and effective for relatively small problems. The use of the latter methods heavily depends on how the partition problem is defined and structured. In particular, the latter methods provide polynomial-time algorithms that are specialised to obtain an optimal solution for well-structured and well-defined problems. However, such methods do not provide algorithms that are effective and efficient across different formalism of goal partition problems. Thus, the exhaustive search method is selected when the problem model is small and does not conform to the form and structure of problems that correspond to linear program integer program and MDP policy synthesis.

Exhaustive search methods can be used to tackle many NP-hard problems that can not be handled using highly efficient algorithms. As the algorithms are usually less effective for problems that exhibit no regular structures. For instance, the travelling salesman problem comprising a large state space is widely considered as an extremely compute-intensive problem that should be tackled by obtaining a good approximation for solutions. The ongoing pattern in advancing the memory size and computation power of hardware has enabled the use of exhaustive search methods to tackle many instances of the travelling salesman problem that was once considered to be intractable [141, 175].

The second method, linear programming, requires that the system-level constraints and optimisation objective can be encoded as a linear programming problem. This method can be used to formulate partition problems that belong to the *deterministic* class [57]. The outcome from solving this class of problems determines with *certainty* (i) the assignment of sub-goals to each component of a system, such that to achieve the system goals. (ii) the cost and utility associated with implementing the sub-goal by each component. The linear program method can handle partition problems that factor uncertainty [101]. The uncertainty may arise to capture the possibility in which a system component may fail to deliver its sub-goal. For instance, the outcome of executing a certain task by a robot may depend on the likelihood of facing some disturbing events such as weather conditions, obstacles in the path, etc. Typically, such events are factored in the goal partitioning model as probability distributions. Each distribution captures the likelihood in which an event may occur. Nevertheless, this thesis focuses on using the linear programming method to handle partition problems that are classified as deterministic. Typically, the model that corresponds to such problems does not factor uncertainty in the sense that the problem inputs are known. Unlike the MDP strategy synthesis method where the problem model accounts for uncertainty in achieving system goals. In general, the linear program method is used to capture partition problems comprising linear decision variables, linear inequalities constraints and a linear objective function.

Many real-world partition problems can be formulated using a linear program. For a distributed SAS comprising a team of mobile robots, the linear programming method can be used to represent partition problems that need multi-robot coordination and task allocation. Such problems may involve tasks such as search and rescue, surveillance, warehouse management, landmine detection can be represented and solved using the linear programming method [134, 171]. These problems share similarities in the characteristic of their tasks. In general, such problems comprise distinct tasks that spread over multiple geographic locations. The multi-robot system needs to collaborate to execute these scattered tasks to satisfy system goals. Examples of tasks derived from those problems can be surveying a square kilometre to detect landmines, or moving some goods from one location to another in a warehouse environment.

The third method uses integer programming for variants of the goal partitioning that can be expressed as an integer programming problem. In particular, the decision variables in a pure

integer programming problem must be discrete (e.g., to which robot a task should be assigned), while the decision variables in a linear programming problem are continuous (e.g., determining the distance each robot should cover). An integer programming problem is referred to as a mixed-integer problem when it contains both discrete and continuous decision variables. Unlike linear programming, an integer programming problem can factor logical constraints on the decision variables. In general, an integer programming problem can include binary decision variables which enable formulating logical constraints. For instance, we can formulate a conditional constraint to ensure that if $robot_1$ is assigned task (A) then $robot_2$ should carry out task (B).

The generic formulation of a partition problem (i,e. presented in Section 3.4.1) provides insight into the scope of goal partitioning problems that can be represented using integer programming. The integer programming method is considered important since many practical situations can be modelled as integer programs [56]. For example, the method can be used to represent allocation problems involving scheduling robots to perform tasks [55] or assigning robots to tasks [128]. Similar to the linear programming method, the scope of using the integer programming method is limited to partition problems that are classified as deterministic.

Integer programming is a known branch of combinatorial optimisation problems which are often referred to as NP-hard optimisation problems [187]. Obtaining a solution in polynomial time for a problem instance of considerable size is known to be a challenge. Typically there is not a single algorithm that can be used to tackle different forms of this problem. Instead, there are various heuristic approaches and approximate algorithms [81] for obtaining in some cases an optimal solution and in others a near-optimal one.

Finally, the fourth method employs MDP strategy synthesis to allocate system goals. This method is feasible if there exists an MDP model that encodes the various possible partitions of system goals. This model exploits the notion of non-determinism to represent the task allocation problem. Unlike the previously mentioned methods (i.e, namely the exhaustive search, linear program and integer program methods), the MDP strategy synthesis method accounts for uncertainty in satisfying system goals.

The MDP strategy synthesis method can be used to handle partition problems that are classified as combinatorial optimisation problems. In general, the MDP strategy synthesis method is able to solve an optimisation problem of this class if the optimisation metrics (i,e. metrics like utility, energy and time) of such problems can be encoded as reward structures. More specifically, the alternative values for the optimisation metrics need to be encoded as parameters of an MDP strategy. In addition, the optimisation problem is solvable if the problem constraints and optimisation objective can be specified as a PCTL multi-objective property over the problem model.

3.7 Summary

This chapter provided a generalised formalisation of the goal partitioning problem for distributed SAS with decentralised control (in Section 3.1), and introduced four methods for solving important classes of this problem.

The classes of goal partitioning problems solved by the four methods are not disjoint. For instance, exhaustive search could potentially be used instead of integer programming or for MDP strategy synthesis if a deterministic strategy is sought, but would be extremely inefficient. To aid the selection of a suitable method, we summarise the advantages and limitations of the proposed methods in Table 3.13.

Table 3.13: Comparison of methods used to partition system goals

Method	**Algorithms/Tools for partition efficiency**	**Probabilistic behaviour**	**Characteristics of contribution amounts**
Linear program	Many available algorithms such as [103, 109]	No	Supports continuous decision variables
Integer program	Some classes of the problem can be solved with polynomial time algorithms	No	Supports only discrete decision variables
MDP strategy synthesis	Multiple probabilistic model checking tools [65, 119]	Supports partial probabilistic behaviour	Supports only amounts encoded as the parameters of an MDP policy

For each of the four goal partitioning methods, a detailed example was provided to illustrate the steps of applying the method. The examples from Sections 3.3.2, 3.4.2 and 3.5.2 are based on multi-robot systems that will also be used (in Chapter 5) to evaluate the theoretical results from this chapter and the nuDECIDE software platform from the next chapter.

CHAPTER 4

DECENTRALISED CONTROL FRAMEWORK FOR SELF-ADAPTIVE SYSTEMS

This chapter describes the nuDECIDE framework for the engineering of the decentralised control software for self-adaptive systems. The framework comprises a decentralised control architecture (presented in Section 4.1), a reusable software platform (i.e., middleware) that uses the goal partitioning techniques from Chapter 3 and reifies the architecture (presented in Section 4.2), and an approach for specialising the components of the software platform for a specific application (detailed in Section 4.3).

4.1 nuDECIDE Architecture

To enable the practical use of the DECIDE philosophy detailed in Section 2.6.1 and of the goal partitioning techniques developed in Chapter 3, we devised the generic nuDECIDE architecture shown in Figure 4.1, a preliminary version of which was presented in [191]. This architecture comprises a fully decentralised hierarchical MAPE-K control instances. In particular, each decentralised control instance comprises;1. a parent **System-level** MAPE-K control loop, which partitions the system goals among the system components under conservative assumptions about the environment, and 2. a subordinate **Component-level** MAPE-K control loop, which ensures that each component achieves its planned contribution to the system goals for as long as these assumptions are met.

Violation of assumptions require a repartition of the system goals by the high-level control loop, and the frequency of these partitions depends on how conservative the assumptions are. Unlike many existing approaches used for developing distributed SASs, nuDECIDE uses formal

analysis within both control loops, with scalability concerns addressed thanks to the hierarchical nature of the architecture. The hierarchical control architecture enables the split of adaptation concerns, which yields smaller formal models to analyse at runtime and subsequently reduces the computation burden. A key ingredient of our approach is the use of *conservative assumptions* in the **Analysis** activity of the system-level control loop. For instance, the analysis activity in a robotic system may assume that a robot travels a distance slower than expected in past missions and consumes more energy. This conservative assumption reduces the need for frequent adaptation among the distributed autonomous system and allows extra *slack* for the analysis activity in the component-level control to handle changes detected in the environment that do not violate the assumptions.

The system-level *Analysis* step on each system component uses formal verification to identify alternative contributions that the component can make for the achievement of the system goals. The execution of this step is infrequent as it is carried out only when a new component joins the system, or the component-level MAPE-K loop can not achieve the component-level goals. Each contribution is associated with *cost(s)* for implementing this contribution. For example, for a robotic team used in a search and rescue mission, each robot can cover different parts of the search and rescue area, and each of these parts is associated with different (predicted) energy consumption, risks, etc. When a component computes new contribution summaries, these contributions are propagated to the *Peer Monitor* activity located on all peer components. Those contributions along with the contributions proposed by *Analysis* instance running on each peer component are passed to the *Plan* activity located on each component system-level control.

Plan activity uses formal techniques to formalise and partition system goals among the distributed components. The use of formal techniques provides guarantees that each component can accomplish its assigned (sub)goals. The *Execution* step is responsible of configuring the component-level MAPE-K in line with the assigned sub-goals. Once the component-level control loop is configured, the *Local Monitor* oversees its operation. The component-level plan (*P*) step synthesises the adaptation plan for the component *Execution* step. When the plan step faces difficulties in synthesising a local adaptation plan, the *Local Monitor* is notified to synthesise a global adaptation plan. Then the *Local Monitor* invokes the *Analysis* step to recalculate the capability summary.

As in [34], the analysis of local capability summary is carried out under conservative assumption. This renders the execution of the system-level Analysis to be infrequent. The *Analysis* step is only invoked when the component plan step experiences difficulties or when a new component joins the system. Furthermore, nuDECIDE can vary how conservative its assumptions are, e.g., it may assume that the robot from our example will consume up to 10% or up to 50% more energy than usual. This enables a wide range of trade-offs between the efficiency with which the components operate, and the frequency with which the component-level MAPE-K loops become unable to synthesise a component-level plan after environmental changes and need to report

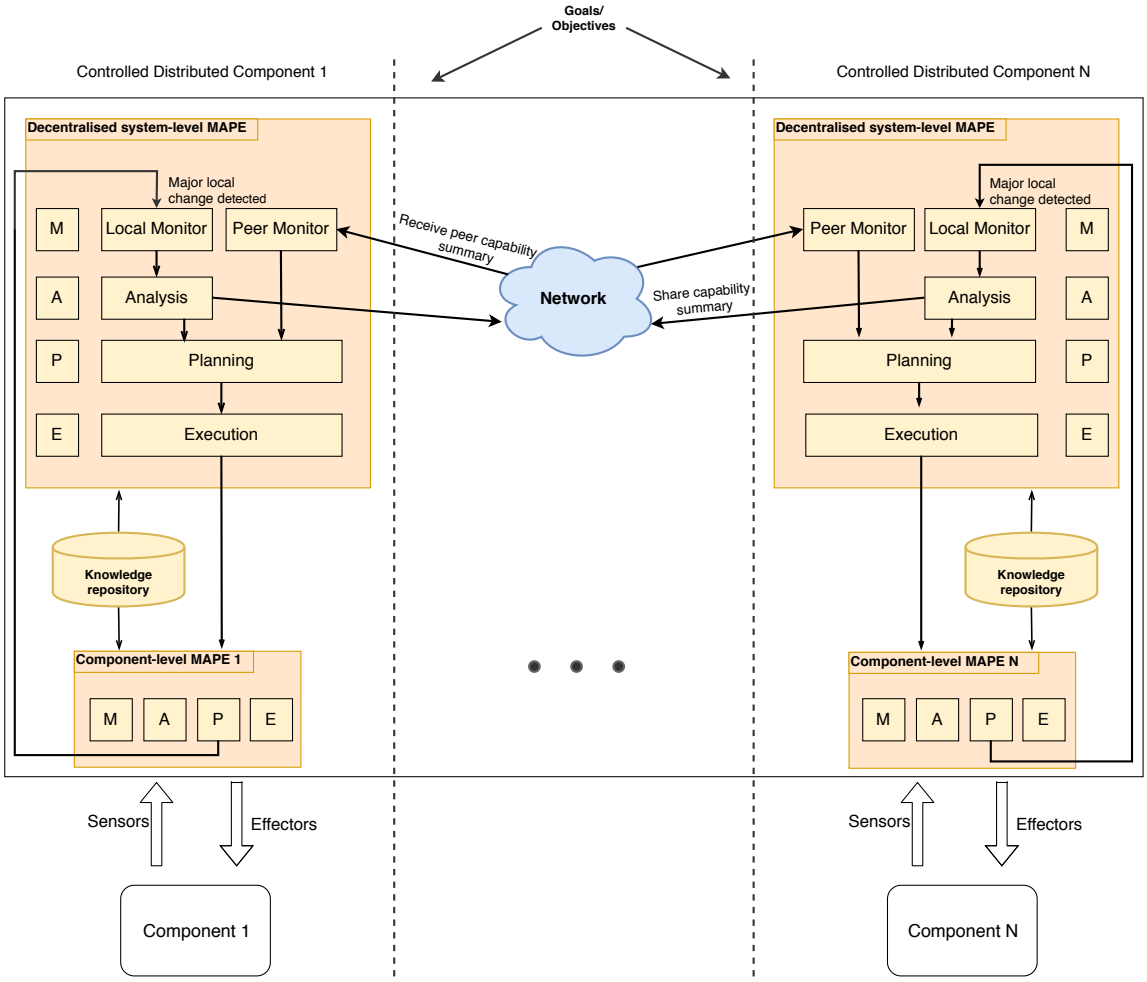

FIGURE 4.1. Fully-decentralised version of the hierarchical-control architecture.

a *major local change* to the Local Monitor. Thus, a robot that assumes up to 50% higher than normal energy use will be less likely to encounter situations where this assumption is violated than a robot that assumes only up to 10% extra energy use; but the former robot would be allocated a smaller part of the search and rescue area in the Planning step than the latter robot.

The use of a fully decentralised system-level MAPE-K loop provides significant benefits such as the absence of single points of failure, and the reduced need for inter-component communication and synchronisation [63]. However, alternative variants of the architecture can use other decentralised MAPE-K loop patterns [184]. In particular, the centralised execution of the Planning has the advantage that the system-level goal partitioning is not redundantly performed by each component, and therefore it can employ techniques that may not yield the same partition each time they are applied (e.g., meta-heuristics as in [83]).

The workflow executed by each component of a nuDECIDE self-adaptive system is detailed in Figure 4.2. This diagram depicts how the stages of the decentralised control are carried out

supported by different types of the *knowledge* underpinning the MAPE-K loop implemented by nuDECIDE:

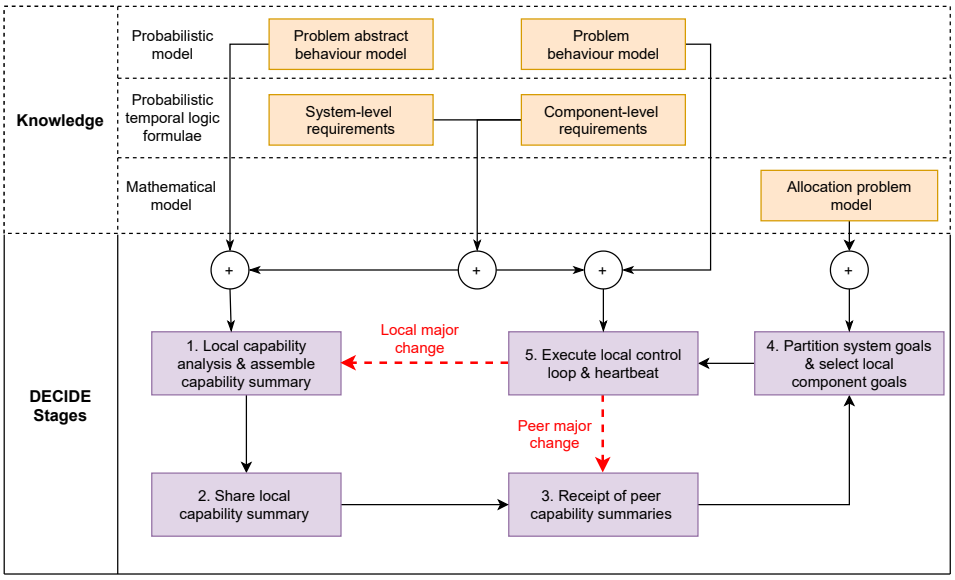

Figure 4.2: Decentralised control workflow executed by each component of a DECIDE distributed autonomous system

Local capability analysis and assemble of capability summary

In this control stage, each nuDECIDE component uses probabilistic model checking at run-time to compute alternative contributions that are relevant for fulfilling the system goals. The computation considers any component-level requirements that might exist, and generates a *local capability summary*, which comprises a finite set of alternative contributions towards achieving the system goals. nuDECIDE approach relies on the use of conservative assumptions about the component's environment to reduce the recurrence of global-adaptation and provide the component with an opportunity to address locally environment changes that do not invalidate these conservative assumptions. For instance, in identifying the contribution of a multi robot system, we may assume that robots execute tasks with higher latency and energy consumption than in previous missions. The pessimistic assumptions reduce the need for frequent global adaptation and provide each robot with an opportunity to handle changes detected in the robot environment that do not violate these assumptions. As a result, those assumptions make the execution of this stage infrequent and reduce the need to reallocate system goals.

nuDECIDE uses a statistical type of estimation referred to as confidence interval [66] to quantify the level of conservativeness that has been assumed when a component capability summary is identified. A capability summary is obtained with an *α-level of confidence* about environment uncertainty. Consider the earlier example of the multi-robot system, if a robot

computes its capability summary with 0.99 level of confidence, the robot becomes less impacted with changes that stem from interacting with environment. As the local control instance that manages this robot would be able to address locally adaptation concerns with a probability of at least 0.99. The more value applied to the confidence level factor (α), the more conservative the assumptions are with regard to environment uncertainty. In general, if an attribute that corresponds to the system requirements is computed with an α-level of confidence, then the local control can fulfill the system requirement that is associated with this attribute, with a probability of at least α.

Definition 4.1. An *α-confidence capability summary* for a nuDECIDE component comprises a finite sequence of tuples containing attribute values, which are computed under *conservative* assumptions. The capability summary of the i-th nuDECIDE component comprises $\{(tuple_1),\ (tuple_2),\ldots,\ (tuple_n)\}$ and the attribute values of each n-tuple takes the following generic form:

$$\underbrace{(attrVal_1,\ attrVal_2,\ldots,\ attrVal_N)}_{tuple_{i,1}} +\ldots+ \underbrace{(attrVal_1,\ attrVal_2,\ldots,\ attrVal_N)}_{tuple_{i,n}}$$

The conservative assumptions enable the local control instance to handle environment uncertainty and achieve these attribute values with a probability of at least α. Since the attribute values are obtained with α-level of confidence, the local control instance can provide guarantee that it can conservatively and simultaneously accomplish these values unless the local control instance of the component encounters disruptive environmental change that violates the assumed range on environment scenarios. The identification of the values of these attributes consider any local constraints that the component must meet to satisfy the component-level requirements. However, the attributes that are associated with the component-level requirements are not included in the capability summary as they are not relevant to the partition of system goals.

Each contribution summary establishes the quality-of-service (QoS) characteristics (e.g., throughput, energy consumption, etc.) that are associated with contributing to the fulfillment of system goals. The executions of local capability stage is infrequent and occurs when a new nuDECIDE component joins the system and receives capability summaries from the controller instances that manage other nuDECIDE components. Also this stage is executed after a nuDECIDE component recovers from a failure and when the managed component experiences partial loss in capability, or disruptive environment change that render the component unable to fulfill system goals. Figure 4.3 illustrates the expected inputs and outputs of this control stage along with the various steps to compute the capability summary of a nuDECIDE component.

In general, a capability summary for a nuDECIDE component is identified through the exhaustive analysis of component configuration space (Cfg) against environment state to find the set of configurations that satisfy the system and component constraints. Then, a component-level objective function which encodes some optimisation criteria, is used to identify a set of

possible alternative contributions. To this end, the local capability analysis control stage requires as an input a set of all possible configurations, which corresponds to all possible combinations of values for the configurable parameters of a nuDECIDE component. This control stage takes as prerequisite, as well, a number of parameters which define the environment state (i.e., parameters that the component can only observes). Then in step (A) as shown in Figure 4.3, an abstract behaviour model for a nuDECIDE component is synthesised from a model template to include the configurable and environment parameters.

This model is used to exhaustively analyse all possible alternative configurations of a component against the environment state to define the behaviour of a component towards satisfying system goals. The model is abstract as it assumes a wide range of environment scenarios and focuses on evaluating attributes that are relevant to the partition of system goals. For instance, in the search and rescue mission described above, the mission may require that the robotic team partitions the areas that need to be searched. Describing the robot's behaviour in executing a search task in each area requires the synthesis of a model for each task. For a robot to establish its contribution summary, the robot needs to perform an exhaustive analysis of its behaviour in executing each task. Performing such analysis at run-time would be infeasible as it introduces a high computation burden and delays in addressing adaptation concerns.

To reduce such computation burden, we may use a single abstract model instead of using a unique model for describing the robot's behaviour in executing each search task. This abstract model assumes a wide range of environment scenarios and can be used to identify the robot set of possible alternative contributions to satisfy system goals (i.e., the system goals in this example is to partition the search and rescue areas) . Typically, an abstract behaviour model is small enough to allow performing an exhaustive analysis of a nuDECIDE component configuration space at run-time, and such analysis is carried out using only modest computational resources.

In step (B), we pass the formalised probabilistic temporal logic formulas, which are used to specify properties that need to be evaluated, along with the synthesised model from step (A) to the model checking engine. Subsequently in Step (C), the model checking engine uses the synthesised model from step (A) to carry out an exhaustive verification of all possible alternative configurations of a nuDECIDE component against all the probabilistic temporal logic formulas to obtain the values of attributes that appear in the system and component requirements. If the values for attributes that correspond to a given configuration satisfy the system and component constraints, then the configuration is considered to be a feasible configuration, in which it can be used to fulfill the system and component requirements. Then in Step (D) of the figure, all the feasible configurations that are identified as a result of verifying the configuration space of a nuDECIDE component are passed to a component-level optimisation function. The optimisation function may encode several QoS metrics, such as energy time and utility, to find the set of optimal configurations that minimise/maximise a single optimisation objective.

Finally, the values of the attributes that appear in the system-level requirements are used to

assemble the possible alternative contributions of a nuDECIDE component using the component set of optimal configurations. Thus, the output of the local capability analysis control stage is a capability summary that contains a set of alternative contributions, where each contribution is associated with its QoS characteristics. These alternative contributions and the value of attributes that appear in the system-level requirements can be used to partition system goals. The execution of this control stage is infrequent and occurs when a nuDECIDE component joins the distributed SAS, and when the component experiences partial capability loss/recovery or disruptive environmental changes.

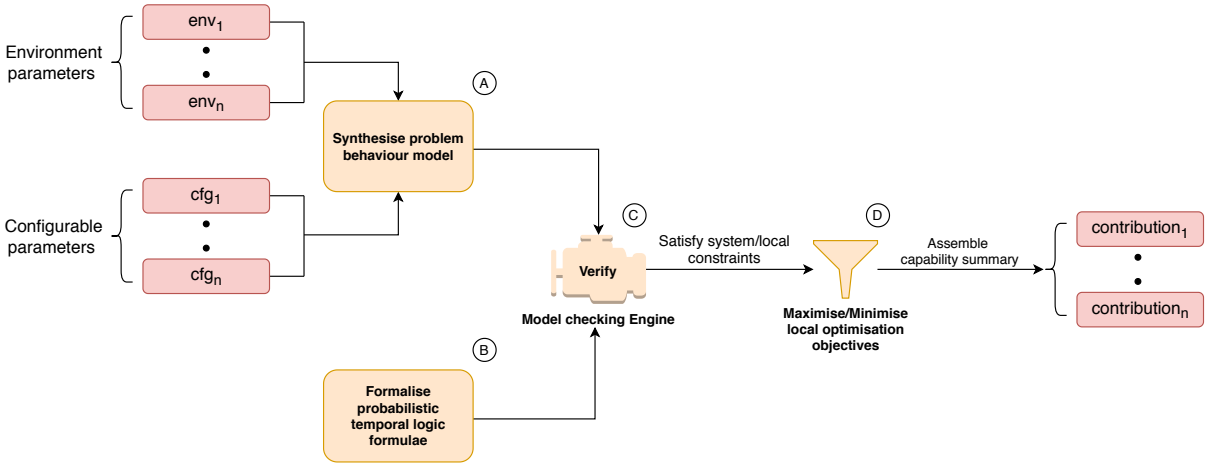

FIGURE 4.3. Process for computing and assembling a capability summary executed by a DECIDE component

Share local capability summary

In this control stage, a nuDECIDE component shares with peer components, the assembled capability summary, which contains all possible alternative contributions and the QoS characteristics associated with achieving these contributions. A capability summary of a nuDECIDE component is expressed as a finite sequence of tuples, where each tuple contains the value of attributes that have the generic form $(attrVal_1, attrVal_2,..., attrVal_N)$. The capability summary is produced under conservative assumptions in which, we assume that a nuDECIDE component can achieve its contributions with α confidence. An α-confidence capability summary corresponds to the level of conservativeness assumed for the environment parameters that are used to produce the capability summary. In general, a capability summary identified with an α level of confidence means that a nuDECIDE component can achieve the contributions associated with this capability summary with a probability of at least α. This control stage is triggered after a nuDECIDE component identifies and assembles its capability summary as part of executing the previous control stage.

Receipt of peer capability summary

As indicated in Figure 4.2, this control stage is executed infrequently when a nuDECIDE component receives capability summaries identified by peer nuDECIDE components as they join the system or experience local major changes (i.e., changes that require a re-partitioning of the system goals, and that are local to a specific component). The execution of this control stage is skipped when a nuDECIDE component experiences *local major change* that requires the component to recompute and assemble its capability summary. This control stage is skipped as the capability summaries of the peer components are unaffected in this case.

Partition System Goals and Select Local Component Goals

Figure 4.4 details how a nuDECIDE system comprising distributed components partition system goals. The objective of this control stage is to partition the system goals into component (sub)goals, such that the system is guaranteed to comply with its goals as long as each component achieves its (sub)goals. As shown in Figure 4.4, a nuDECIDE component uses as an input its capability summary along with summaries computed by other components to synthesise at run-time an *allocation problem model* that formally describes the system goals and the possible alternative contributions of the components involved in the the partition (i.e., such model typically only include the capability summaries of active components that can contribute towards achieving system goals). Depending on the used formal technique, the synthesis of an allocation problem model may factor as input environment parameters that describe the behaviour of a nuDECIDE component in fulfilling system goals.

nuDECIED approach is not perspective about goal allocation technique used to partition system goals. As demonstrated in Examples 3.3.2, 3.4.2 and 3.5.2, the choice of which technique to use for partitioning system goals depends on the characteristics of the systems requirements. Thus, the approach assumes a formalism that is problem specific that can represent the task allocation problem. Also, such formalism must include the system constraints and optimisation objective that need to be factored in when system goals are partitioned. System constraints are included to represent for instance the limitation in a given component resource. For example, we may include a constraint to place a threshold for the amount of energy or time a component can use to achieve system goals. An optimisation objective is used to encode a metric of choice such as utility, time or energy consumed, which are factored in to identify the optimal allocation of system goals.

As seen in Figure 4.4, we assume that the partition of system goals (i.e., the *partition goals* process) is carried out by the DECIDE controller of every component and using an efficient and deterministic technique such as the methods introduced in Chapter 3 to select the local goals of each nuDECIDE component. The selected local component goals represent the *contribution level agreement* (CLA) of a component, chosen such that the system is guaranteed to comply with its goals as long as each component achieves its CLA.

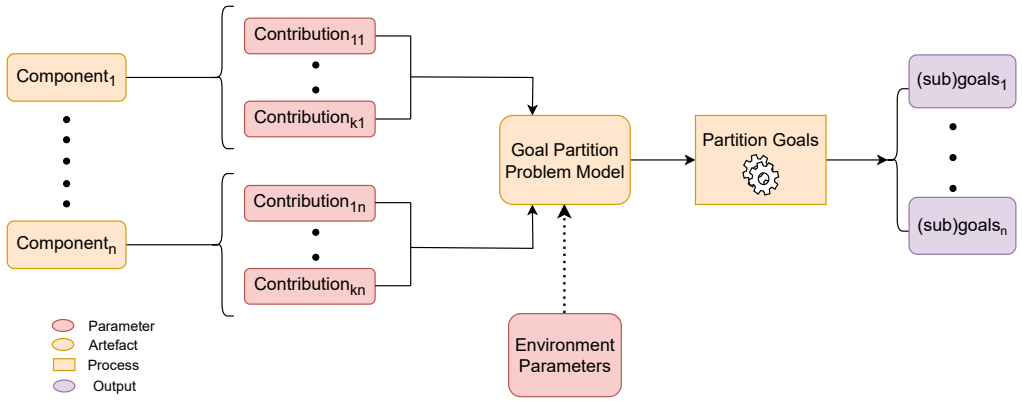

Figure 4.4: Process for goal partitioning of system goals executed by each component of a DECIDE distributed SAS

Execution of local control loop and heartbeat

Once a nuDECIDE component selects its CLA, the local control loop ensures that the component comply with the selected local component goals. In particular, the local control loop uses probabilistic model checking to analyse a continually updated *problem behaviour model* that captures the state of the component and its environment. When the nuDECIDE component selects its local component goals, the local control loop synthesises a problem stochastic model inline with its CLA. The local control loop may also update when needed the thresholds that are encoded in the system-level constraints. This update enables the local control loop from verifying if the component is complying with its CLA.

In general, this control stage is the predominant control stage in nuDECIDE control workflow, as the local control loop and the maintenance of a low-overhead system *heartbeat* are the activities that are executed most of the time. Infrequently, events such as disruptive environmental changes (e.g., significant workload increases) or loss of component capabilities (e.g., the loss of a sensor) render a DECIDE local control loop unable to achieve the local goals; or the disappearance of a heartbeat indicates the complete failure of a peer component. These events (shown by dashed arrows in Figure 4.2) are termed major changes, and their detection by the local control loop is followed by the computation and selection of new sets of local goals for the (remaining) DECIDE components.

4.2 nuDECIDE Software Platform

To ease the adoption of the nuDECIDE architecture in practical applications, we developed a reusable software platform (i.e., middleware) that implements all the application-independent functionality of the nuDECIDE architecture and control workflow. Figure 4.5 shows the components of this software platform, which we implemented in Java. It also highlights the high-level

architecture of a nuDECIDE team devised using our framework. The figure shows the two options for forming the team members that are deployed to execute the mission. In the first option, the team comprise several mobile ActivityBot robots with a high-level system controller on a laptop that coordinates the robots. For the second option, the team consists of multi-robot simulator instances with a high-level system controller on a laptop that coordinates the robot simulator instances. Instrumenting a robot simulator for a specific robot team application requires specialising a JAVA built software component that comprises $ManagedComponentSimulator.java$ class. In using either of the deployment options, there is a two-way communication between the robots (i.e., or the robot simulators) and laptops as they interchange information related to the mission.

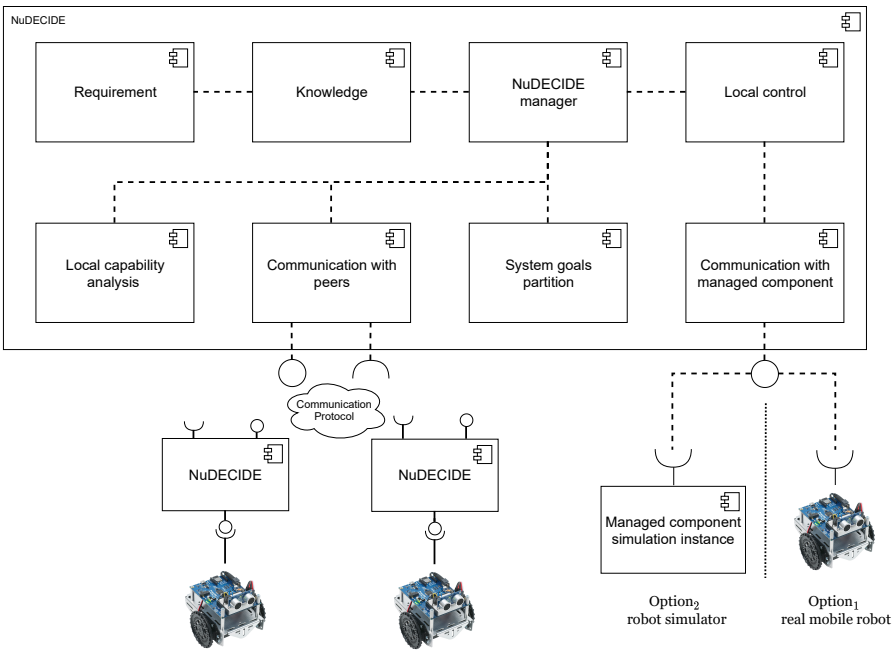

Figure 4.5: Each robot is controlled by a nuDECIDE instance. nuDECIDE manager oversees the overall execution. nuDECIDE components could be specialised based on application-specific requirements

In the oil pipeline inspection mission described in Section 3.3.2 of Chapter 3, the robot will be sharing the status of its environment parameters, such as the degradation in its sensors. While, the running nuDECIDE control instance performs high-level actions such as requesting the robot to move to a specific position or altering the robot speed sp_i. The nuDECIDE framework has been used to develop several case studies from the oil pipeline inspection and waste management domains. In the remaining part of this section, we illustrate how the nuDECIDE components are specialised to devise the high-level controllers for the pipeline inspection application. The lab-based testbed comprising the team of ActivityBot and the robot simulator instances are described in Chapter 5 as part of the evaluation for the applications that use either of the deployment

options.[1] Specialising the main nuDECIDE components (Java classes) shown in Figure 4.5 are essential for instrumenting nuDECIDE for a specific robot team application. The nuDECIDE control workflow shown in Figure 4.2 illustrates the activities which these components combine to implement. In the following, we describe the components used to devise the nuDECIDE controller for pipeline inspection application and where relevant we include the number of the nuDECIDE workflow stage associated with each component.

nuDECIDE manager. The java class $MainPipeInspection.java$ has the responsibility to control the mission and ensure that the robots are executing and achieving their tasks. This class oversees the robot operations and executes the nuDECIDE control stages shown in Figure 4.2. This class also has the responsibility to execute the reallocation of tasks if needed and make sure that the mission's overall goals are achieved.

Local Capability Analysis (Stage 1) : The $RobotLocalCapabilityAnalysis.java$ class employs probabilistic model checking to identify alternative contributions that the robot can make towards the system goals that are encoded in the *Allocation Problem* model as part of Knowledge layer. To establish the robot contributions, the class analyses recursively through probabilistic model checking the robot's capabilities that are encoded in the robot *Abstarct Behaviour* model. Typically, an object of the $RobotConfiguration.java$ class encodes robot configuration elements, and defines a robot capability. In particular, the class object defines the value for the configuration elements sp_i, $p_{1i}$and p_{2i}, which determines the speed of a robot and the sensors that are used to execute the inspection task. In general, the configuration elements act as parameters in an abstract behaviour model, which describes the robot's behavior during an inspection task.

To assess the robot capabilities, the probabilistic model checking engine is provided with an abstract behaviour model that encodes robot configuration elements and the environment state. The environment state is encoded in an object of the $RobotEnvironment.java$ class, which defines the environment elements (p_{1i}^{retry}, p_{2i}^{retry}). Where elements p_{1i}^{retry} and p_{2i}^{retry} define the probabilities that the measurements taken by sensors p_{1i} and p_{2i} respectively, are of insufficient accuracy. The nuDECIDE controller obtains the values of these environment elements at run-time from each robot. The result from the capability analysis summarizes robot's alternative contributions, which include QoS metrics associated with each contribution such as the time and energy required to perform an inspection task and the total energy available in the robot to execute inspection tasks. Such metrics are encoded in an object of $RobotCapabilitySummary.java$ class and transmitted to peer nuDECIDE controllers. The capability analysis is performed infrequently when a robot joins/rejoins the system or after partial capability loss/recovery.

[1]Lack of access to the mobile robots testbed due to COVID-19 restrictions made it impossible to evaluate all case studies using real robots.

Communication with Peers (Stage 2): The java class $CommunicateWithPeers.java$ has the responsibility to share with peer nuDECIDE controllers the contribution summaries identified as part of the capability analysis (Stage 1), and execute the heart beat mechanism, which traces the changes experienced by the peer controllers (Stage 3). The class implements an application specific communication interface that allows the exchange of data between the nuDECIDE controllers. In the pipeline inspection application, the transmission of data is carried out using broadcast communication protocol [148], however any reliable communication protocol can be used to exchange data between the nuDECIDE controllers. To share the contribution summaries, the nuDECIDE controller serialises objects of $RobotCapabilitySummary.java$ class that contain the robot's alternative contributions, and transmits these contributions to peer nuDECIDE controllers.

Subsequently, the nuDECIDE controller uses the contributions that has been identified by its local capability analysis along with the alternative contributions received from peer nuDECIDE controllers (in stage 3) to find an optimal partition/re-partition of the inspection tasks. Also, the $CommunicateWithPeers.java$ class has the responsibility to execute the heart beat mechanism. Executing the heart beat mechanism entails using a dedicated execution thread that sends periodic alive messages to peer nuDECIDE controllers and traces messages received from the peer controllers to assess that each peer controller is functioning as usual. If the heart beat mechanism identifies incidents such as a peer robot joins the team or has experienced a major change or failure, that necessitates the re-partition of inspection tasks, the mechanism interrupts the execution of the local control activity (stage 5), which is implemented using $RobotLocalControl.java$ class. The execution of the communication with peers stage may be skipped in the rare event, when the team of robots comprises a single robot only.

Allocation of System Goals (Stage 4): The $LinearSelectionMethod.java$ class has the responsibility to synthesise the Allocation Problem model and execute the goal allocation algorithm to partition inspection tasks. In the case of pipeline inspection application, the partition of system goals entails implementing an API for an open-source solvers called *OR-Tools* [146] for combinatorial optimization problems and encoding the linear optimisation problem. The optimisation problem comprises linear decision variables, linear inequalities constraints and a linear objective function. The problem is solved using the linear optimiser to obtain the optimal values for the decision variables. The capability analysis is performed infrequently when a robot joins/rejoins the system or after partial capability loss/recovery.

The execution of the linear allocation method in $LinearSelectionMethod.java$ is triggered when the robot identifies and shares its contribution summaries (Stage 1) or when a peer robot shares its capability summaries (Stage 3) . As a result, each robot's controller performs the partitioning algorithm to create an identical partition of system goals. The aim is to partition inspection tasks and identify each robot‘s local goals such that the multi-robot system is guaranteed to comply with its goals as long as each robot achieves its local goals. For example, the team

of robots complies with its system goals when each robot performs its inspection task(s) within the time limit allocated to complete a duty round.

Local controller (Stage 5): The $RobotLocalControl.java$ class implements a local control loop that employs probabilistic model checking to ensure that the robot complies with the selected local goals. The probabilistic model checking is carried out using a continually updated stochastic models of the robot and its environment. The probabilistic model checking engine is provided with a stochastic model that defines the robot's behavior in line with the robot's assigned local goals.

Similar to the local capability analysis, the behaviour model that encodes robot configuration elements and the environment state and the model is included as part of Knowledge layer. The nuDECIDE controller obtains the values of these environment elements at run-time from the robot. The class is responsible of executing a low-overhead heartbeat mechanism to ensure the liveliness of the underlying managed robot. Infrequently, events such as disruptive environmental changes (e.g., significant degradation in sensor performance) or a sudden robot failure render a nuDECIDE local control loop unable to achieve the local goals. Also, the execution of the local control loop is interrupted when a peer robot shares its capability summaries or experiences a complete failure.

4.3 nuDECIDE Engineering Approach

This section describes the four-step approach that needs to be carried out to use the software platform presented in the previous section to implement a nuDECIDE self-adaptive system. The description of the approach is illustrated using the pipeline inspection robotic system introduced in Section 3.3.2 as a running example.

4.3.1 Behaviour model development

In the first step of the nuDECIDE approach, the software engineers need to devise the probabilistic models that describes the stochastic behaviour of a nuDECIDE component. Describing the methods used to obtain these models is beyond the scope of this chapter. Using inaccurate models to reason about a nuDECIDE component through probabilistic model checking may lead to incorrect self-adaptation actions. There is a great body of knowledge on approaches to devise such stochastic models. For instance, a common approach is to relay on domain experts to devise a stochastic model for a system by analysing the system's logs to elicit its behaviour models [85]. Another approach is to use model to model transformation to devise stochastic models [13]. Finally, numerous examples of developing such models are available from the research literature in application domains including service-based systems [36, 178], software product lines [86, 87], robotic system controllers [2], web applications [135, 136] and others.

A nuDECIDE control instance typically uses the probabilistic model checking to analyse stochastic models that describe the behaviour of a component and its environment. These models can be any of the Markov model variants introduced in Chapter 2. The choice of which model variant to use depends on the characteristics of the requirements that need to be analysed.

However, the model used must capture the behaviour of a nuDECIDE component, in line with the sub-goals that can be assigned to the component. This type of models is typically probabilistic and focuses on configuration/environment parameters that influence the values of attributes that correspond to the component-level requirements. However, these type of models do not need to abstract out environment parameters as these models only represent a proportion of system goals, so they are typically small enough to enable an efficient verification. We assume that the verification techniques used by the system-level and component-level controllers are based on the same modelling paradigm. This is essential as the analysis stage of the system-level controller plans for the assumed range of environmental scenarios what a component-level controller will achieve of system sub-goals.

For the pipeline inspection self-adaptive system from our running example, this step involves devising the DTMC model from Figure 3.1. This model is described in detail in Section 3.3.2.

4.3.2 Identification of configurable and observable parameters

In this step of the nuDECIDE approach, the software engineers have to define the parameters that influence the behaviour of a nuDECIDE component towards satisfying its requirements. Mainly, a stochastic model includes two types of parameters. An *observable* parameter is the first type of parameters, which is used to capture the environment state. While the second type are the *configurable* parameters, which represent the parameters that a nuDECIDE controller can alter to address changes observed in the environment parameters. Cfg_i denotes all possible combinations of values for the configurable parameters of nuDECIDE component i and the environment space Env_i is the set of all possible combinations of values for the environment parameters of the same component. Typically, a component i reacts to changes observed in the environment parameters by selecting a configuration from all possible combinations of values for the configuration parameters Cfg_i.

For the pipeline inspection self-adaptive system from our running example, this step involves devising the configuration parameters for the DTMC model from Figure 3.1.

Example

The configuration parameters for robot i from the running example in Section 3.3.2 are $p_{i1} \in [0,1]$, $p_{i2} \in [0,1]$, and $sp_i \in (0,\ sp_i^{max}]$. Where p_{i1}, p_{i2} specify the probability for robot i to use either of its two sensors to perform the inspection tasks. Typically the probability for robot i to use its second sensor p_{i2} is $1-p_{i1}$.Thus, the configuration space for robot i to execute in inspection task

of type k is

$$Cfg_{ik} = \underbrace{[0,1]}_{p_{i1}} \times \underbrace{[0,1]}_{p_{i2}} \times \underbrace{(0,\ sp_i^{max}]}_{sp_i}$$

The environment parameters for robot i are its probabilities $p_{i2}^{retry}, p_{i2}^{retry}$ of having to repeat inspection task as the obtained reading is of insufficient quality. The environment space Env_i for a given pair of sensors that are specialised to perform an inspection task of type k is the set of all possible combinations of values for the environment parameters and is defined as follows:

$$Env_{ik} = \underbrace{[0,1]}_{p_{i1}^{retry}} \times \underbrace{[0,1]}_{p_{i1}^{retry}}$$

For the m-robot system to complete their mission successfully, they should comply with system-level and component-level requirements. To evaluate these requirements, we use a probabilistic model checking engine to verify the functional and QoS attributes associated those requirements. Functional attributes may include the type of inspection tasks that the robot can perform for a given configuration. QoS attributes are concerned with Qos characteristics such as timeliness, energy consumption, etc. with which these tasks are executed. The evaluation is carried out using the probabilistic model checking, which analyses a model that describes a component's behaviour and encodes the state of its configurable and observable parameters against a probabilistic temporal logical formula Φ, as described in Section 2.6.1.

4.3.3 Specification of system-level and component-level requirements

In this step, the component-level requirements need to be specified in a probabilistic temporal logic appropriate for the model(s) used to establish the component attributes as shown in Section 3.3.2 (i.e., PCTL for DTMCs and MDPs, or CSL for CTMCs). Additionally, the system-level goals (constraints and optimisation objective) need to be expressed in a formalism suited for the goal partitioning method used by the distributed SAS under development.

For instance, for the robotic system from the running example, we used a DTMC to model the relevant behaviour of individual robots, so the component requirements were specified in PCTL, as detailed in Section 3.3.2, while the system-level requirements were specified as constraints and an optimisation objective that, taken together, defined a linear programming problem (see Section 3.3.2).

4.3.4 Specialisation of the software platform

In this step, the software engineer needs to define Java classes that specialise the abstract classes of the nuDECIDE software platform. This development process is described in Section 4.2, and involves defining Java classes that extend the key components of the nuDECIDE platform, including the $SystemGoalsPartition.java$ abstract class. For the pipeline inspection case study, the software engineer has to specialise this class to define the abstract functions' behaviour shown

in Figure 4.6. In particular, the $LinearSelectionMethod.java$ class includes the *synthesisePartitionModel* method, which enables encoding the linear decision variables, linear inequalities constraints and linear objective function that are used to formulate the linear partition problem model. While the *execute* function includes the linear programming solver, which uses the encoded partition problem model to identifies the optimal allocation of system goals.

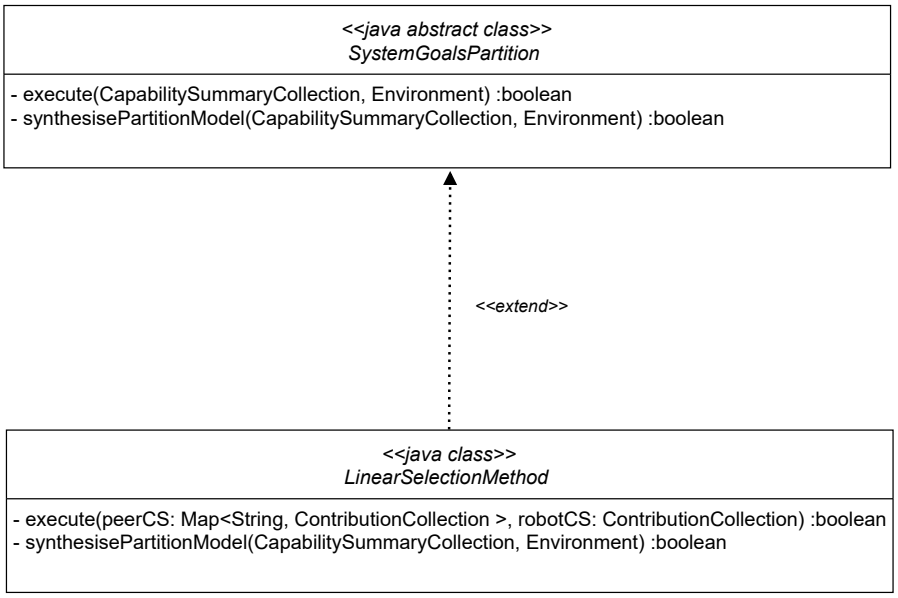

Figure 4.6: A class diagram for extending the $SystemGoalsPartition.java$ class

4.4 Summary

This chapter presented the nuDECIDE architecture and software platform, and described the engineering approach for developing a nuDECIDE self-adaptive system by specialising the software platform.

CHAPTER 5

EVALUATION

In this chapter, we describe the extensive experiments carried out to evaluate the goal partitioning methods introduced in Chapter 3 and the nuDECIDE framework introduced in Chapter 4. We performed this evaluation for a range of case studies, using both real mobile robots (for the linear programming method from Section 3.3, Chapter 3) and simulation (for the integer programming and MDP strategy synthesis techniques from Sections 3.4 and 3.5 in Chapter 3)[1]

The chapter is organised as follows. First, Section 5.1 describes the evaluation of the nuDECIDE framework when using the linear programming method for system-level goal partitioning. This evaluation was carried out using a testbed comprising a team of three mobile robots. Next, Sections 5.2 details the evaluation of the software platform for a simulated multi-robot mission that used the integer programming goal partitioning method. Finally, Section 5.3 presents the evaluation of the platform combined with the MDP policy synthesis for the goal partitioning. Two different scenarios were assessed for this method, namely when the MDP encoded only the nondeterminism associated with the goal partitioning, and when the MDP encoded both this nondeterminism and probabilistic aspects of robots execution of tasks.

5.1 Evaluation of Linear Programming method

5.1.1 Research questions

From reviewing the literature, numerous realistic robotic team missions exist [7, 58, 107] whose requirements can be formalised in such a way that the linear programming method can be used. The experiments carried out to evaluate this method aimed at assessing whether:

[1] Lack of access to the mobile robots testbed due to COVID-19 restrictions made it impossible to evaluate the last two methods using real robots.

1. Given a robotic team mission whose requirements can be formalised as a linear programming problem, nuDECIDE can use linear programming method to successfully adapt the decentralised robotic team to a range of disruptions comprising the following:

   - robots failing completely/leaving the team;
   - new/recovered robots (re)joining the team;

   where by successful adaptation we mean the following:

   - after a period of time since the disruption, the tasks of the mission are partitioned to the robot team with consideration of modified robot team members and/or modified robot capabilities, and in line with the system requirements;
   - with nu-DECIDE appropriately configured, the period of time from the previous bullet point is "short" (i.e., within seconds).
   - nuDECIDE with linear programming method can operate with acceptable overheads for at least small teams of robots.
   - nuDECIDE is configurable so that optimality and overheads of the solution can be traded-off for one another, i.e. there is a way for the user to select values for the nu-DECIDE configuration parameters so that the robots synchronise more frequently and therefore handle disruptions faster if needed, but this means higher overheads. Conversely, the user can configure nu-DECIDE to operate with lower overheads, but this means a longer time to recover from disruptions.

2. nu-DECIDE with linear programming method can operate with acceptable overheads for at least small teams of robots.

To carry out this assessment, we formulated the research questions (RQs) summarised below.

RQ1 (Effectiveness) Can nuDECIDE with linear programming method handle different patterns of disruptions from item 1 above successfully?

The experiments performed to answer this research question aim to illustrate the effectiveness of nuDECIDE in addressing adaptation concerns for distributed self-adaptive systems. To demonstrate the effectiveness, we present several end-to-end scenarios in which the system from the case study experiences a number of synthesised events over a simulated period of time. Consider, for example, the case study of pipeline inspection, which is described in Section 3.3 of Chapter 3. We have conducted several experiments that demonstrate how several real mobile robots are deployed to simulate performing gas inspection tasks. In order to assess the effectiveness of the decentralised control loop, we simulate several mission scenarios that include a number of synthesised events over a simulated period of time. These events demonstrate different failure patterns that the multi-robot system may experience when deployed in a real mission. For

example, $robot_2$ fails at time instance 120, recovers at time instance 250, etc. These experiments aim to build confidence that the approach is effective in general, and the decentralised control loop deals with adaptation concern without undermining the satisfaction of system goals and component goals. These experiments include a pattern of failures that necessitate the repartition of system goals or in some other scenarios, the synthesis of a local adaptation plan when the control instance that manages a robot can address an adaptation concerns locally. Thus, the aim is to build confidence that the system addresses adaptation concerns and provides the necessary assurance that the components are achieving their goals. These events differ in the way they are synthesised when simulating the missions and are categorised into two types.

The first type of synthesised events includes handcrafted events scripted using prepared scenarios. In this type of synthesis, the system from a case study experiences several patterns of failures that are predetermined over a simulated period of time. The second type of event synthesis comprises scenarios that include random events. The randomness covers the timing of which an event is introduced and the type of failure pattern that is simulated. The failure patterns differ according to the characteristics of the system being tested.

RQ2 (Scalability) How does the overhead of using nu-DECIDE with linear programming method increase as the number of robots in the team grows?

For evaluating the above research questions, we carried out several experiments using a 2.5 GHz Intel Core i7 MACBOOK Pro computer with 16GB memory, running Mac OS X 10.14. All experiments that required the partition of system goals using the linear programming method, we implemented an API for an open-source solvers called OR-Tools [146], which enables encoding and solving linear programming problems.

5.1.2 Case study: pipeline inspection

Using the testbed described in Section 5.1.3, which includes real mobile robots and a testing track designated for each robot. We conducted two sets of experiments aimed at evaluating nuDECIDE. The first set includes experiments to assess the effectiveness of nuDECIDE in addressing adaptation concerns. The second set aims at assessing the scalability of nuDECIDE. In the second set of experiments, we examine the effect of increasing the number of robots on the ability of nuDECIDE to deal with systems of different sizes. In particular, the experiments related to the scalability concern focus on the partition phase of the system goals, since the analysis accompanying the partitioning process is performed at a system level and is considerably affected by the increase in system size.

The first set of experiments comprises ($m = 3$) real mobile robots, while the experiments concerned with scalability simulates the use of $3 \leq i \leq 34$ robots to imitate the steady increase in the number of robots from one experiment to another. The results of the second group of experiments are obtained through simulations due to the lack of sufficient space to carry out experiments

System characteristic	**Parameter space**	**Parameter description**
Speed of robot$_i$	$sp_i \in (0,\ sp_i^{max}]$	A *configurable* parameter determines the speed in moving between task regions
Sensors per robot$_i$	$m_i \in \{2,\ 4\}$	robot$_i$ is *deployed* with $m_i \in [2,\ 4]$ on-board sensors and each pair of sensors in robot i is specialised to perform a specific type of task from types $\{taskTyp_m,\ taskTyp_p\}$.
Sensor probability	$p_{ij} \in [0,1]$	p_{ij} is *configurable* parameter specifying the probability for robot i to use the j-th sensor to perform the inspection task

Table 5.1: Characteristics of the i-th robot deployed to perform mission tasks

and the limited number of real robots available for testing. In general, the experiments differ in terms of the number of robots m used.

Table 5.1 illustrates the characteristics of the ensemble of robots used in these experiments. The robots' characteristics may differ from one robot to another, as is the case in the number of sensors the robot has and the type of tasks that a sensor can perform. All experiments were carried out on a 2.5 GHz Intel Core i7 Macbook Pro computer with 16GB memory, running Mac OSX 10.14.

5.1.3 Robotic testbed

This section introduces the Parallax ActivityBot robots that are used to simulate the execution of the missions described in the earlier presented case studies. This section also reviews the different components that make up the robot that were used during the experiments. Also, the section describes the program, that is developed for the purpose of this research, and used to control robots.

ActivityBot Hardware

The Propeller Activity Board is the most important component of an ActivityBot robot, which enables an 8-core Propeller P8X32A-Q44 micro-controller to control all the connected sensors and components [98]. The micro-controller comprises eight 32-bit processors, called cogs that enable the robot to execute simultaneous tasks such as using a sensor to gather an environment reading while traveling to a destination. An ActivityBot robot comes equipped with wheels that are connected to High-Speed Servos, which allow the robot to move in any direction. These Servos can continuously rotate the wheels in two different directions (e,g, bidirectional way) and move these wheels at varying speeds ranging from 0 to up to 180 RPM [98]. ActivityBot robots are equipped with four IR sensors that enable each robot to track the path and follow it back and forth. An ActivityBot robot can be programmed a language derived from C/C++ called Propeller

C [98] and includes a metallic chassis that enables mounting the IR sensors, and a breadboard for connecting the sensors and other circuitry to the CPU. Each robot is programmed to navigate the testing track using data collected from its sensors, which enables the robot to Simultaneously determine its location and the regions it must visit to carry out its tasks.

Also, each robot is mounted with a communication module called WiFi module WX ESP8266, as shown in Figure 5.1. The communication module enables the robot to send and receive two classes of communication messages. The first class enables the robot from sending periodic *alive* messages indicating the state of the robot and includes simulated readings for changes detected in the robot's environment. The *alive* messages enable the running instance of nuDECIDE control from deducing whether the robot has failed by examining the difference between sequential alive messages. Also, this class of messages includes information that enables the controller from reasoning about the robot's behavior and address adaptation concerns when the behaviour violates the constraints and objectives imposed on the robot in carrying out its tasks. The second class of messages enables the robot to receive commands sent from the nuDECIDE control, which specify the tasks that the robot must perform and configure the speed that the robot must adhere to when heading to the regions, where tasks are executed.

Figure 5.1: WiFi communication module

ActivityBot Software

The program for controlling the ActivityBot is illustrated in the activity diagram shown in Figure 5.2. This software is deployed in all the ActivityBot robots that are used to execute the mission described in 3.3.2. The software comprises two parallel tasks that allow the robot to execute mission objectives. The first task enables the robot to communicate with the nuDECIDE controller that coordinates its action. The second task involves executing a line following algorithm that enables the robot to detect where it's about from the testing track route.

In the first task, an ActivityBot robot has to establish a serial connection using a WiFi module that enables communication with the nuDECIDE control instance running on payload computer. If such communication is established, the robot has to send periodic messages within some time window. These messages typically include environment readings that define the probability of how accurate is the readings provided by the robot's sensors. Also, the messages demonstrate

to the nuDECIDE controller that the robot does not experience a fetal failure. The second task is executed in parallel with the first task. It involves running a nested loop that enables the robot from realising its position from the testing track and define to which region/segment of the pipeline the robot should move.

controls the activityBot

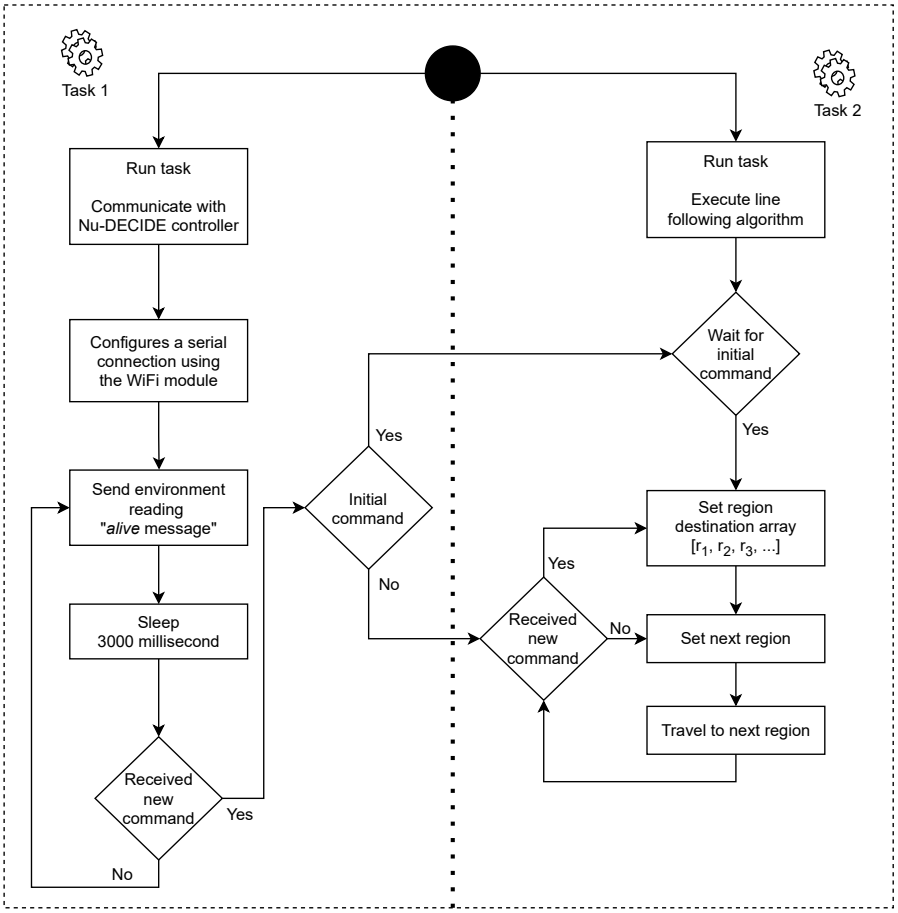

Figure 5.2: Activity diagram for the software running on the ActivityBot robots

5.1.4 Experimental results

5.1.4.1 RQ1 (Effectiveness)

The testbed provided as part of the development of our approach along with case studies allow for easy experimentation with goal partition techniques. For evaluating the effectiveness of nuDECIDE, we used this testbed, which comprises ($m = 3$) real mobile robots and a testing track designated for each robot. Figure 5.3 illustrates the testing track, where each consists of ($k = 8$) tasks and a path that links these tasks in sequential style. The testbed simulates a refinery environment containing tasks of two different types. $tasTyp_m$ and $tasTyp_p$ are the first and

Robot i	**# deployed sensors**	**Type of task(s)**
robot_1	$m_1 = 2$	$tasTyp_m$
robot_2	$m_2 = 2$	$tasTyp_p$
robot_3	$m_3 = 4$	$\{tasTyp_m, tasTyp_p\}$

Table 5.2: Characteristics of sensors deployed in the i-th robot

second types of tasks. In particular, tasks $\{1,2,3,4\}$ are of the first type, while tasks $\{5,6,7,8\}$ are of the second type. In this section, we will review the results obtained from analysing a typical

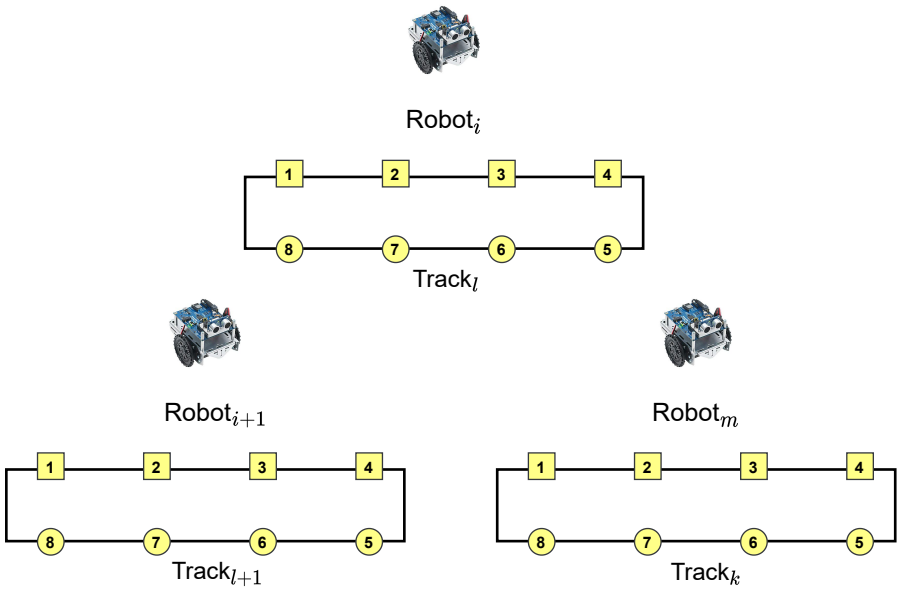

Figure 5.3: Testbed used to experiment with m-robot system

mission comprises ($m = 3$) robots that partition the tasks shown in Figure 5.3. Table 5.2 shows the sensors that each robot is equipped with, which in turn determines the type of tasks a robot is able to execute. In general, the results obtained from experiments are categorised according to the method used in synthesising the events. In particular, there are predefined events and randomly synthesised events. Robots must fulfil the mission constraints and objectives, outlined in Table 5.3, when they are deployed in scenarios involving any of these events. We set the upper bound of constraint (l_2) to 15 minutes. This constraint defines the time allocated to complete a duty round that the m-robot system must adhere to when carrying out their tasks. Similarly, we set the upper bound of constraint (l_3), which defines the maximum energy that robot $i \in \{1,2,3\}$ may use during a duty round, to 180, 200, and 260 joules respectively. In the following, we present the results obtained from deploying Activitybot robots in two missions to simulate robots' use to perform gas inspection tasks in a refinery plant environment. The scenario of the first mission

Requirement Type	**Details**	**Description**
constraint (l_1)	$\forall\ 1 \leq i \leq n\ .\ x_i e_i \leq E_i^{max}$	The energy required by robot$_i$ to execute all tasks should be less than the energy available for the robot per a duty round of the mission
constraint (l_2)	$\forall\ 1 \leq i \leq n\ .\ x_i t_i \leq t$	The time required by robot$_i$ to execute all the assigned tasks should be less than 15 minutes, which is the time available to complete a duty round
Local objective	miminise $\sum_{j=1}^{n} w_1 \times e_{ij_i} + w_2 \times t_{ij_i}$	If multiple configurations satisfy constraints l_1, l_2 and l_3 and then the i-th robot should use the configuration that minimises the cost function
System objective	maximise $\sum_{i=1}^{n} u_i x_{ij_i}$	If multiple configurations satisfy constraints l_1, l_2 and l_3 then the multi-robot system should maximise the covered area such that to reduce the area covered by a human operator

Table 5.3: Characteristics of mission constraints and objectives

includes predetermined events (e,g., handcraft events), while the second scenario contains events that are randomly synthesised. Events that occur in both scenarios simulate patterns of failures, such as robot failure, sensor degradation, etc.

Predetermined events. Figure 5.4 shows the timeline for a mission execution scenario consisting of three robots collaborating to achieve mission objectives. It shows the period between the emergence of events and addressing the adaptation concerns raised by these events. The figure also illustrates the execution of the nuDECIDE stages by these robots over 700 seconds (roughly 12 minutes) of the simulated time period.[2] The circled numbers 1,2,3,4 represent the execution of nuDECIDE stages by the control instance running on each robot. Figure 5.5 depicts an activity diagram that describes 1-4 stages of nuDECIDE. Table 5.4 illustrates the characteristics of the events derived from the mission execution scenario and affect the multi-robot system. In the following, we review the time moments (t) for these events and describe some of nuDECIDE controllers' primary operations running on these robots:

- ($t \approx 0$s). At system startup, nuDECIDE control instance running on each robot executes the local capability analysis stage and share contribution summary with peer controllers. After sharing the summary, nuDECIDE control waits for a specified time window to receive contribution summaries from peer controllers. Then each nuDECIDE control instance executes the partition of system goals stage. Subsequently, this stage defines each robot (sub)goals based on the alternative contributions proposed by the nuDECIDE control

[2]Note that the timings from the experiment assume very short task execution times for convenience. In a real-world scenario, performing this mission may well require many hours.

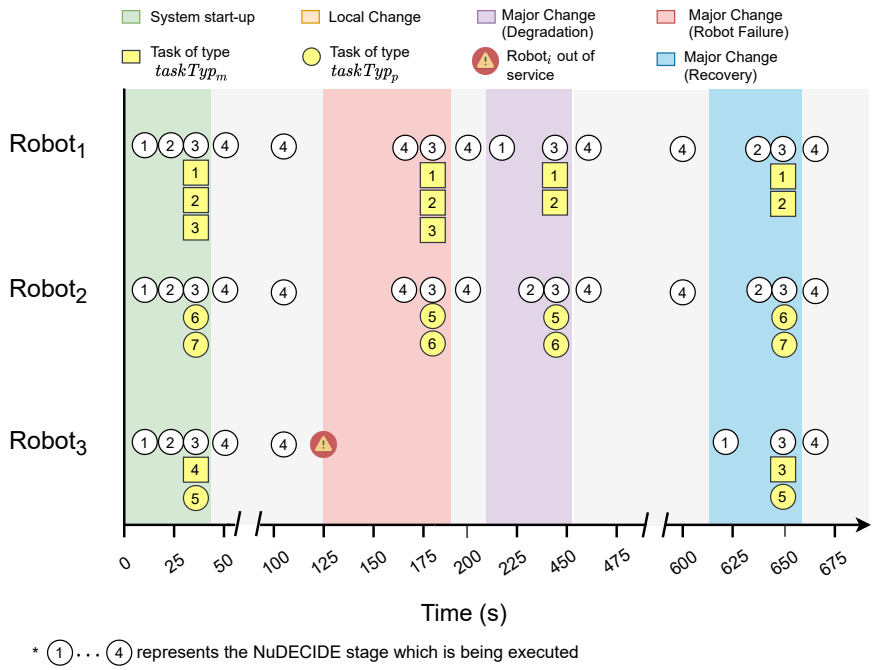

Figure 5.4: Execution of nuDECIDE stages 1–4 for a three-robot deployed in a mission involves handcrafted event

Robot (i)	**Sensor** (p_{ij}^{retry})	**Event period (start:finish)**	**Type of event (*failure (f), degradation (d), recovery (r)*)**
robot$_1$	p_{11}^{retry}	206:255,	d 65%
robot$_3$	p_{31}^{retry}, p_{32}^{retry}, p_{33}^{retry} and p_{34}^{retry}	125:192, 619:654,	f, r

Table 5.4: Events introduced in the mission scenario

instances running on each robot. The execution of the partition stage identifies the CLAs of robot $i \in \{1,2,3\}$ per task type as follow:

- robot$_1$ = $tasTyp_m${2.83, 71.41, 14.7}
- robot$_2$ = $tasTyp_p${2.4, 69.02, 14.97}
- robot$_3$ = $tasTyp_m${1.165, 37.97, 6.75}, $tasTyp_p${1.397, 40.93, 8.24}

The CLA defines the robot's contribution to each type of tasks that the robot has the specialised sensors to perform. In particular, the CLA specifies for each the amount of distance the robot must cover for each type of gas inspection tasks. The CLA also determines how much energy and time each robot can use during a duty round to perform its tasks. In this mission, a task is defined as the meter distance a robot has to inspect with its sensors.

- ($t \approx 125$s). Robot$_3$ experiences a failure renders the robot unable to continue fulfilling its CLA. As a result, the robot stops sending its periodic heartbeat messages and loses contact with its peers (robot$_1$ and robot$_2$). After a period of time (approximately 35 seconds), the heartbeat mechanism running on robot$_1$ and robot$_2$ detects that robot$_3$ is missing and subsequently interpret the execution of local control stage. Then, robot$_1$ and robot$_2$ re-execute the partition of system goals stage to define the CLA for each robot, which is as follows:

  - robot$_1$ = $tasTyp_m\{2.83,\ 71.41,\ 14.7\}$
  - robot$_2$ = $tasTyp_p\{2.4,\ 69.02,\ 14.97\}$

  As Figure 5.4 depicts, robot$_1$ has to perform the task of inspecting the presence of methane gas in which the robot must cover a distance of 2.83 meters to inspect the leakage of methane gas. While robot$_2$ needs to inspect 2.4 meters of the pipeline to ensure there is no leakage of propane gas.

- ($t \approx 206$s). Robot$_1$ undergoes a major local change where Sensor 1 suffers from a significant degradation of service. As it is unable to meet its CLA, it computes and shares with peer its new contribution summary. The new robot's CLA provides less contribution to system requirements as the sensor affected by the degradation takes less time when taking readings. The new CLAs established for robot$_1$ and robot$_2$ are as the following:

  - robot$_1$ = $tasTyp_m\{1.69,\ 58.71,\ 14.92\}$
  - robot$_2$ = $tasTyp_p\{2.4,\ 69.02,\ 14.97\}$

- ($t \approx 619$s). As robot$_3$ has recovered after experiencing a total failure, the robot calculates and notifies peers of its contribution summary, and thus initiates a new CLA re-negotiation which results in the following CLAs:

  - robot$_1$ = $tasTyp_m\{1.69,\ 58.71,\ 14.92\}$
  - robot$_2$ = $tasTyp_p\{2.4,\ 69.02,\ 14.97\}$
  - robot$_3$ = $tasTyp_m\{1.165, 37.97, 6.75\}$, $tasTyp_p\{1.397, 40.93, 8.24\}$

Random events. For the second type of experiments, which is performed to answer the research question RQ1, the experiment was carried out using $m = 3$ real-robots that were deployed in a mission scenario that included randomly generated disruptive events. The mission scenario includes disruptions events such as degradation of a robot sensor, total failure of a robot or one of its sensors. It is worth mentioning that randomness in generating events is not limited to the type of the generated event but includes the timing during which an event occurs. There is a time gap between 20 and 200 seconds between the successive events in synthesising these events. In the

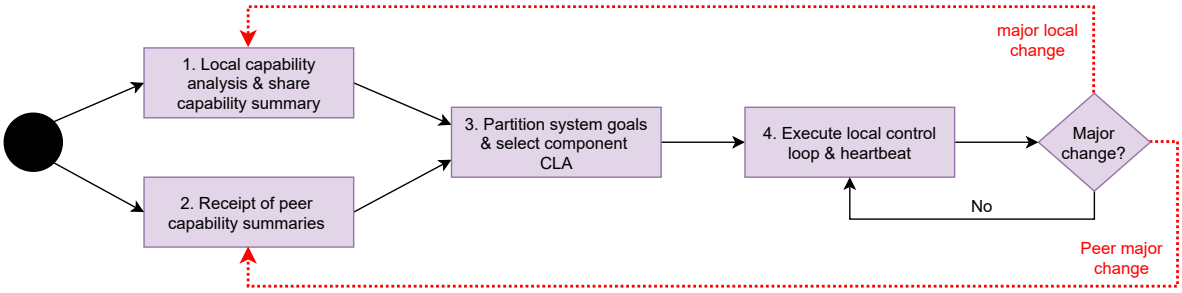

Figure 5.5: Workflow of a decentralised nuDECIDE controller

following, we review the results of one of these experiments, which cover a period of 725 seconds, in Figure 5.6. Table 5.5 demonstrates the characteristics of the randomly generated events which derive from the actual mission execution. The system parameters for this experiment contain the values mentioned in Table 5.9, and the description of the scenario and adaptation actions performed by the robots is provided below:

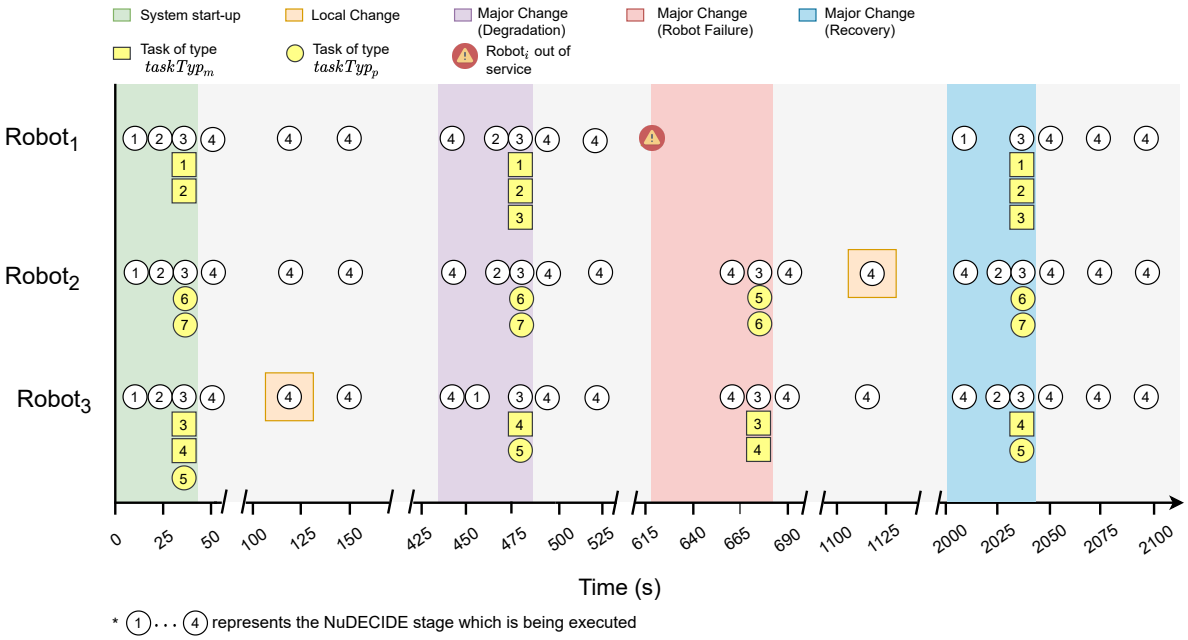

Figure 5.6: Execution of nuDECIDE stages 1–4 for a three-robot deployed in a mission involves random event

- With in the first 60s of the experiment, the three robots are forming into a team. At the system startup stage the nuDECIDE control instance running on each robot computes its contribution summary and shares its this summary with peers. After sharing the summary, nuDECIDE control waits for a specified time window to receive contribution summaries from peer controllers. Then, each control instance uses linear programming method to

Robot (i)	**Sensor** (p_{ij}^{retry})	**Event period (start:finish)**	**Type of event (*failure (f), degradation (d), recovery (r)*)**
robot$_1$	p_{11}^{retry}	360:409, 626:668	f, r
robot$_1$	p_{12}^{retry}	360:409, 626:668	f, r
robot$_2$	p_{21}^{retry}	489:509,	d 22%
robot$_3$	p_{31}^{retry}	104:122, 212:261,	d 12%, d 28%

Table 5.5: Random events introduced in the mission scenario

formulate and partition system goals, in which the method rigorously selects the local goals for each robot. Thus, the execution of the partition stage establishes the initial CLAs for robots 1, 2 and 3 per each task type as follow:

  - robot$_1$ = $tasTyp_m${1.52, 38.56, 14.84}
  - robot$_2$ = $tasTyp_p${1.87, 53.76, 14.89}
  - robot$_3$ = $tasTyp_m${1.76, 50.8, 8.9775}, $tasTyp_p${0.73, 21.38, 5.2}

- Between time instant t = 42s and t = 106s, the robots carry out their tasks as allocated, and regularly exchange heartbeats, run the local control loop.

- At approximately time instant t = 125s, robot$_3$ experiences a major local change where Sensor 1 suffers from a slight degradation of service 16%. To continue achieving its CLA, the robot changes its configuration to an alternative configuration that satisfies its CLA. In particular, the robot reduces its speed from 4 cm/s to 2 cm/s and changes its sensor configuration $p_{3,2} = 0.75$ to relay more on Sensor 2 in obtaining measurements. The new configuration of the robot consumes more energy to carry out its tasks, but the robot still achieves its local goals.

- Between time instant t = 212s and t = 261s, robot$_3$ undergoes a major local change where Sensor 1 suffers from a significant degradation of service. In particular, The accuracy of the sensor readings degrades by 28%, which renders the robot unable to meet its CLA. As a result, the robot notifies its peers by computing and sharing with its new contribution summary. By notifying its peers, the robot initiates a new CLA re-negotiation which results in the following CLAs:

  - robot$_1$ = $tasTyp_m${1.52, 38.56, 14.84}
  - robot$_2$ = $tasTyp_p${1.87, 53.76, 14.89}
  - robot$_3$ = $tasTyp_m${0.82, 33.82, 5.83}, $tasTyp_p${0.92, 25.79, 6.34}

The new CLA for robot$_3$ shows that the robot contributes less than before in using its sensors to make measurements along the methane pipeline, while it contributes more to making more measurements along the propane pipeline.

- At time instant $t \approx 360$s, robot$_1$ suffers from a total failure, which makes the robot unable to satisfy its CLA and sending periodic heartbeats to its peers. In return, robot$_2$ and robot$_3$ discover, after a short period of time, about 25 seconds, that robot$_1$ is missing. Thus, the remaining active robots re-execute the partition of system goals control stage and establish the following CLAs:
  - robot$_2$ = $tasTyp_p${1.87, 53.76, 14.89}
  - robot$_3$ = $tasTyp_m${1.34, 54.46, 13.56}
- Between time instant $t = 489$s and $t = 509$s, robot$_2$ experience a local change, where the accuracy of measurements provided by Sensor 2 degrades by 22%. The robot alters its configuration by adjusting the probability ($p_{2,1}$), which makes the robot rely more on Sensor 1 for taking the necessary measurements along the propane pipeline. The new configuration requires the robot to consume more energy to perform the readings, but it meets the robot's CLA.
- At approximately time instant $t = 626$s, robot$_1$ re-joins the system after experiencing a complete failure. The robot notifies its peers, by establishing and disseminating its capability summary. Thus, the robot initiates a new CLA selection, which results in the following CLAs:
  - robot$_1$ = $tasTyp_m${1.52, 38.56, 14.84}
  - robot$_2$ = $tasTyp_p${1.87, 53.76, 14.89}
  - robot$_3$ = $tasTyp_m${0.82, 33.82, 5.83}, $tasTyp_p${0.92, 25.79, 6.34}

5.1.4.2 RQ2 (Scalability)

To assess how a nuDECIDE system that uses linear programming method to partition system goals, can scale with systems with different sizes. We performed a set of experiments and varied the number of robots in each experiment. The number of robots from these experiments is between 2 and 42 robots. The aim from varying the size is to compare the CPU time taken by linear programming solver to partition system goals. As shown in Figure 5.7, the time required for the goal partitioning grows approximately linearly with the number of robots, starting at under 0.0002s for $m = 3$ robots and reaching above 0.0008 seconds for the largest system we considered (with $m = 42$ robots). This shows that the approach is scalable to considerable robot team sizes, when linear programming method is consider for partitioning system goals.

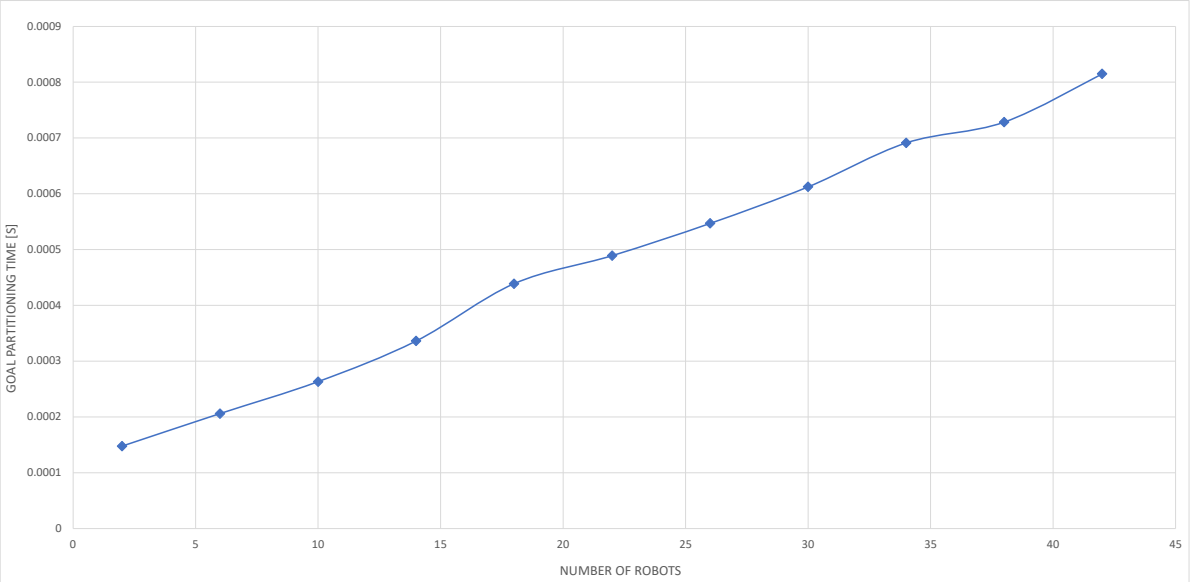

Figure 5.7: nuDECIDE scalability analysis for the use of linear programming method to partition system goals

5.1.5 Discussion

The experimental results from Sections 5.1.4.1 and 5.1.4.2 show that the hypothesis underpinning our evaluation of linear programming method were valid. More specifically, the results from 5.1.4.1 provide evidence that a system comprising a team of real-robots enhanced with self-adaptive capability using nuDECIDE can address adaptation concerns at runtime. The multi-robot system has demonstrated resilience when faced with disruptive changes such as a degradation in sensor measurement accuracy and a total robot failures. In particular, the results indicate the nuDECIDE with the use of linear programming method, has managed to effectively re-partition the system goals, in the case where the disruption can not be handled by the local control instance, or when the multi-robot system realises that one of its robots is missing.

During all these disruptions, the system has proven that it is able to re-partition its goals within a relatively short period of time, thus making the system restore the ability to achieve its goals. For instance, the multi-robot system managed to handle a sudden degradation in a robot's sensor that renders the robot unable to satisfy its CLA. The nuDECIDE controller instance that coordinate this robot computed and shared robot with peer its new contribution summary. Subsequently, the multi-robot system initiated the re-negotiation of selecting a new CLA to restore compliance with system requirements. Those results provide evidence that nuDECIDE can effectively support the dependable adaptation in a system comprising distributed components.

The pipeline inspection case study described in Section 3.3.2 is chosen to illustrate that nuDECIDE can use linear programming method to successfully adapt the decentralised robotic team. In particular, the case study comprises a realistic robotic team mission that includes

requirements that can be formalised using the linear programming method. We believe this case study includes requirements that share characteristics with many of the requirements that appear in many realistic robotic team missions such as [7, 58, 107]. The aim of implementing this case study using the real multi-robot system is twofold.

The first aim is to demonstrate how effective is the use of linear programming method in a team of real-mobile robots that have been enhanced with self-adaptive capabilities using nuDECIDE. Two experiments have been implemented to demonstrate the effectiveness of nuDECIDE that uses the linear programming method to successfully adapt the decentralised robotic team to a range of disruptions. These experiments include a number of synthesised events over a simulated period of time. The second goal is to illustrate that nu-DECIDE with linear programming method can operate with acceptable overheads for systems of different sizes. A set of experiments were implemented to measure the CPU time taken by the linear programming solver to partition system goals. The experiments differ in terms of the size of the goal partitioning problem, where the state space of the problem has increased by adding two robots from one experiment to another. The aim of carrying out these experiments is to illustrate that for a given robotic team mission whose requirements can be formalised as a linear programming problem, nuDECIDE can operate with acceptable overheads.

Moreover, the case study demonstrates that nuDECIDE can capture the behaviour characteristics of the robotic system along with the characteristics of the goal partitioning problem. However, further research is required to assess the applicability of nuDECIDE with linear programming method to a number of additional domains. In particular, we need to examine if nuDECIDE is able to capture the behaviour characteristics of systems from other domains such as IoT and transportation. Also, we need to inspect if the linear programming method can capture the goal partitioning problem from these domains.

In addition, the pipeline inspection case study has been implemented using nuDECIDE framework, which relies on Prism as the model analysis engine of choice. The requirements that have been considered as part of the case study are limited to those which may be represented using discrete-time Markov chains. We envisage that nuDECIDE framework can support the analysis of requirements for systems from other domains if these requirements can be represented using Markovian models. The implementation of the pipeline case study provides an insight into the scope of distributed SASs that can be developed using the nuDECIDE framework. More specifically, the implementation illustrates the characteristics of the modelling techniques that can be used to capture the system behaviour and the goal partitioning problem.

Type of event	Description
Robot failure (F)	robot_i is experiencing a total failure
Battery degradation (D)	A sudden degradation in robot i battery level
Probability Increase (I)	An increase in the probability p_{full} that defines the likelihood in which robot i may encounter the n-th sub-task ($subT_{2,n}$) (i.e., this sub-task requires that a robot may be able to collect a full recyclable bin)
Robot recovery (R)	robot i has joined/(re)joined the team or recovered its capability

Table 5.6: Events introduced in the mission scenario

5.2 Evaluation of Integer Programming method

5.2.1 Research questions

From reviewing the literature, numerous realistic robotic team missions exist [7, 58, 107] whose requirements can be formalised in such a way that the integer linear programming method can be used. In this section, we demonstrate several experiments that aim at assessing the effectiveness and scalability of nuDECIDE when integer programming method is used to formalise and partition system goals. In particular, evaluating the effectiveness in using the integer programming method to formalise the requirements of a given a robotic team mission entails asserting that the nuDECIDE can successfully adapt the decentralised robotic team to a range of disruptions. While, the aim form assessing the scalability is to ensure that nu-DECIDE with integer programming method can be used to partition system goals with acceptable overheads for at least small teams of robots. To carry out this evaluation, we formulated the research questions (RQs) summarised below.

RQ1 (Effectiveness) Can nuDECIDE with integer programming method handle different patterns of disruptions successfully?

To evaluate the effectiveness, we carried out experiments that aim at illustrating how a robotic team that is enhanced with self-adaptive capabilities using nuDECIDE can use integer programming method to successfully adapt the decentralised robotic team to a range of disruptive events. The experiments were performed using a simulated multi-robot missions whose characteristics and requirements are found in Section 3.4.2. The simulated multi-robot system comprises ($m = 3$) robots that are deployed for waste collection in a public park environment. The experiments comprise an end-to-end scenarios that include a number of disruptive events over a simulated period of time. Table 5.6 shows the different patterns of disruptive events that were seeded in these experiments.

The experiments differ in terms of the method used to synthesise events that are included

in the end-to-end scenarios. To assess the effectiveness we performed two experiments. The first experiment comprises an end-to-end scenario that include predetermined events. Whereas, the second experiment includes randomly generated events. The aim from these experiments is to build confidence that the nuDECIDE decentralised controllers can deal with the impact of different types of failure.

RQ2 (Scalability) How does the overhead of using nu-DECIDE with linear programming method increase as the number of robots in the team grows?

To assess the scalability, we carried out a set of experiments aimed at examining the impact of increasing the number of simulated robots on the ability of nuDECIDE to deal with systems of different sizes. In particular, the experiments related to the scalability concern focus on the goal partitioning control stage, since the analysis that takes place at this control stage is performed at a system level and is considerably affected by the increase in system size.

For evaluating the above research questions, we carried out several experiments using a 2.5 GHz Intel Core i7 MACBOOK Pro computer with 16GB memory, running Mac OS X 10.14. All experiments that required the partition of system goals using the integer programming method, we used an interface from the JAVA programming language to a linear and nonlinear integer programming solver called SCIP [78], which is used to formulate and solve the partition problem. This library was used to encode the models for the partitioning problem. These models are used when executing the goal partitioning stage of the nuDECIDE decentralised control.

5.2.2 Case study: waste management

To answer the research questions from the previous section, we considered the multi-robot waste management application introduced in Section 3.4.2. In this application, $m > 1$ mobile robots are used to collect both garbage and recyclables from $n \geq 1$ physically distributed waste collections locations in a public park environment. The full details of the application are provided earlier in the thesis, so this section focuses on describing the two sets of experiments that we ran in simulation in order to address our research questions.

5.2.3 Waste management simulator

A decentralised discrete-event simulator was implemented in Java to enable the experiments detailed in this section. An instance of this simulator was run for each of the m mobile robots from the system. A key component of this instance was the nuDECIDE software platform that was specialised for the waste management application by following the engineering approach described in Section 4.3.

5.2.4 Experimental results

5.2.4.1 Experiments for research question RQ1

For this research question, we carried out two different types of experiments, each of which considered a robot team comprising $m = 3$ robots. The former type of experiment involved creating a synthetic scenario in which the three-robot system had to dynamically adapt in order to cope with a set of predefined disruptions occurring at predefined times. The purpose of this experiment was to establish whether the expected adaptations would be performed by the robots. Depending on the type and severity of the introduced pattern of failure, the expected adaptation may take the following form:

- Assuming that a disruptive event has affected the i-th robot and the local control loop of that robot can address this adaptation concern locally. The robot has to successfully adapt after a period of time since the disruption, and modify its behaviour (i.e., configuration) to continue achieving its CLA;
- If the disruptive event requires that m-robot system partition/(re)partition the tasks of the mission. The multi-robot system successfully adapt after a period of time since the disruption, and the tasks of the mission are partitioned to the robot team with consideration of modified robot team members and/or modified robot capabilities, and in line with the system requirements outlined in Section 3.4.2;
- with nuDECIDE appropriately configured, the period of time from the previous bullet point is “short” (i.e., within seconds).
- nuDECIDE with integer programming method can operate with acceptable overheads for at least small teams of robots.

The values of the system parameters for this experiment are shown in Tables 5.7 and 5.8. While, the experimental results are shown in Figure 5.8 (which covers an 800-second system mission[3]) and are detailed below:

- Within the first 60s of the experiment, the three robots are forming into a team, and the nuDECIDE control instance running on each robot computes a disseminates its contribution summary. Then, a nuDECIDE controller waits for a short period time ($time\_window_2$) which is Table 5.7, to allow the opportunity to synchronise the start timing between controllers and avoid the situation in which a nuDECIDE control instance executes the partition control stage twice as it has started before others. Then each nuDECIDE control instance uses integer programming method to formulate and solve the partition problem. The execution

[3]Like for the robotic-team application from Section 5.1, small execution times were assumed for the tasks performed by the robots. In a real-world scenario, this type a mission is likely to take a much longer time to complete, e.g., several hours.

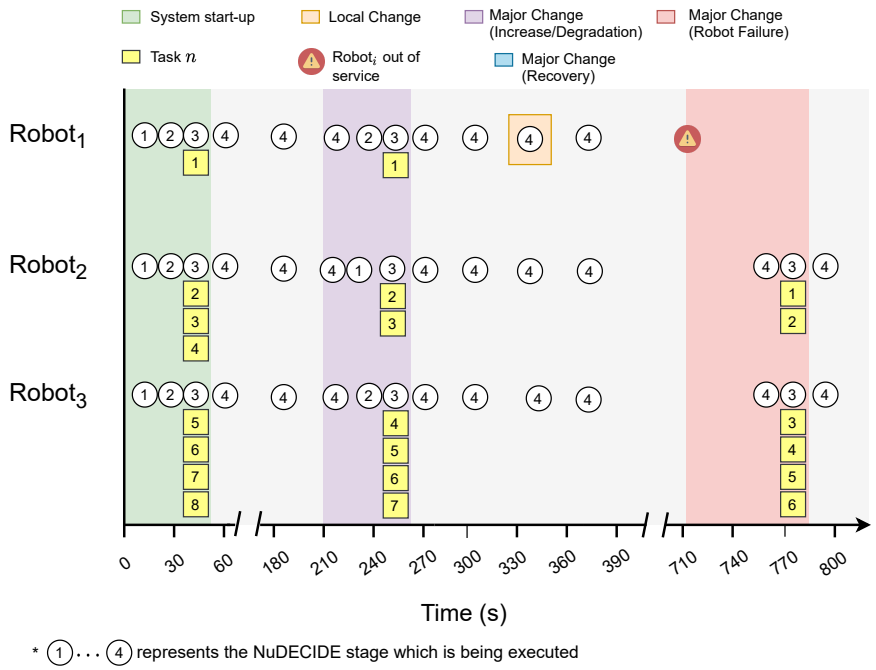

Figure 5.8: Execution of nuDECIDE stages 1–4 for a three-robot deployed in a mission involves predetermined event

of the partition stage identifies CLAs of robot $i \in \{1,2,3\}$. A CLA specifies the collections tasks assigned to each robot along with determining the amount of energy and time (e_i, t_i) a robot can use to perform tasks. The initial CLAs are defined as follow:

- robot$_1$ is assigned collection tasks (1) and must consume energy and time less than (352.28, 310.61).
- robot$_2$ is assigned collection tasks (2, 3, 4) and should consume energy and time less than (1029.96, 940.83).
- robot$_3$ is assigned collection tasks (5, 6, 7, 8) and must consume energy and time less than (987.12, 1070.6).

- Between time instant $t = 60$s and $t = 210$s, the robots carry out their tasks as allocated, and regularly exchange heartbeats, run the local control loop.

- At approximately $t = 210s$, robot$_2$ experiences a significant increase by 75% in the probability (p_{full_2}), which defines the likelihood that the robot would encounter a full recyclable bin. As a result its local control loop can not comply with the robots CLA, and the robot computes and shares to its peers an updated contribution summary. Then, the multi-robot system re-execute the partition of system goals control stage to define the CLA for each robot, which is as follows:

  - $robot_1$ is assigned collection tasks (1) and must consume energy and time less than 352.28, 310.61.
  - $robot_2$ is assigned collection tasks (2, 3) and should consume energy and time less than 858.3, 940.41.
  - $robot_3$ is assigned collection tasks (4, 5, 6, 7) and must consume energy and time less than (987.12, 1070.6).

- Between time instant t = 270s and t = 330s, the team of robots perform their tasks as normal, and regularly exchange heartbeats, run the local control loop.

- At time instant $t = 332s$, $robot_1$ experiences a local change where the total energy available for the robot to achieve its CLA has degraded by 15%. To continue achieving its CLA, the robot reduces its speed from the value 3 to 1 and adjust the probability ($p_{1,2}$) of collecting non-full bin to 0.6 (i.e., the probability of performing sub-task ($subT_3$)).The new configuration reqires that the robot to consume more time in executing its allocated task, however the robot can still meet its CLA.

- Finally, at time instant $t = 712s$, $Robot_1$ experiences a total failure, which renders it unable to fulfil its CLA. As a result of this failure, the robot stops sending periodic messages indicating that it operates as usual. In return, the peers of the robot ($Robot_2$ and $Robot_3$) lose contact with the robot and, after a short period of time, about 25 seconds, begin to re-negotiate the partition of system goals. This short period of time is configurable as part of the parameters from Table 5.7 that define the parameters for the mission's characteristics. The $time\_window_1$ parameter specifies the period of time during which the robot must wait for a heartbeat message from its peers before concluding that they are missing. The new CLAs established for the remaining two robots are as follow:

  - $robot_2$ is assigned collection tasks $1,2$ and should consume energy and time less than 858.3, 940.41.
  - $robot_3$ is assigned collection tasks $3,4,5,6$ and must consume energy and time less than 987.12, 1070.6.

These results provide evidence that the robotic team that is enhanced with self-adaptive capabilities using nuDECIDE can use the integer programming method to successfully adapt the decentralised robotic team to a range of disruptive events such as a sudden increase in a robot's workload and total failure of a robot. More specifically, the results illustrate that if the disruptive event requires the multi-robot system to re-negotiate the partition of system goals as a robot could not achieve its CLA, the affected robot notifies its peers, and a new CLA is selected for each robot in the system. Also, the results demonstrate that if a disruptive event that can be handled by local adaptation occurs, the affected robot changes its configuration to restore its compliance with

Parameter	**Description**	**Value**
n	Number of waste collection tasks	8
$distance$	Amount of distance between the consecutive tasks	180 meter
$time\_window_1$	Amount of time between sending the frequent alive messages (i.e., the frequency of the heartbeat)	20 seconds
$time\_window_2$	Amount of time for the frequency of executing the local control loop	10 seconds
α	Amount of confidence used to compute the capability summary	0.90
$time\_per\_round$	Amount of time available for a duty round	1200 seconds

Table 5.7: Parameters defining the mission characteristics

its CLA. These remarks provide evidence that nuDECIDE can support dependable adaptation as the results indicate that nuDECIDE can handle effectively different pattern of failures.

For the latter type of experiment carried out to answer research question RQ1, the simulation involved having the three-robot system dealing with a similar scenario, but with the adverse events generated randomly, such that:

1. the scenario included random event types covering all types of adverse events listed in Table 5.6;

2. the timing of the events was also randomly chosen, in such a way that there was a gap of between 20s and 200s between events.

We report the results of one such experiment, covering a period of 2000 seconds, in Figure 5.9. The system parameters for this experiment had the values reported in Table 5.9, and the description of the scenario and adaptation actions performed by the robots is provided below:

- Within the first 60s of the experiment, the multi-robot system is forming into a team, and the nuDECIDE control instance running on each robot establishes and shares its contribution summary with its peers. Then, the nuDECIDE control instance waits for a specified period time, to receive contribution summaries from peer controllers. Then the nuDECIDE control instance running on each robot, initiate the the partition of system goals control stage. This control stage uses integer programming method to formulate and solve the partition problem. The execution of the partition control stage results in the CLAs:

  - $robot_1$ is assigned collection tasks (1, 2, 3) and must consume energy and time less than (980.88, 1009.77).

Parameter	Description	Value
Parameters of robot$_1$		
E_1	Amount of energy available per duty round	1200 Joules
B_1	Amount of total energy available for the mission purpose	6000 Joules
sp_1	A variable for the speed of a robot in moving between task locations	$sp_1 \in \{1, 2, 3, 4\}$ m/s
$r_{1,1}$	The rate for executing sub-task ($subT_1$)	0.0023
$r_{1,2}$	The rate for executing sub-task ($subT_2$)	0.0012
$r_{1,3}$	The rate for executing sub-task ($subT_3$)	0.001
$e_{1,1}$	The rate of energy consumption for executing sub-task ($subT_1$)	0.44 joules/second
$e_{1,2}$	The rate of energy consumption for executing sub-task ($subT_2$)	0.51 joules/second
$e_{1,3}$	The rate of energy consumption for executing sub-task ($subT_3$)	0.38 joules/second
$t_{1,1}$	The rate of time consumption for executing sub-task ($subT_1$)	0.019
$t_{1,2}$	The rate of time consumption for executing sub-task ($subT_2$)	0.024
$t_{1,3}$	The rate of time consumption for executing sub-task ($subT_3$)	0.012
Parameters of robot_2		
E_2	Amount of energy available per duty round	1150 Joules
B_2	Amount of total energy available for the mission purpose	5750 Joules
sp_2	A variable for the speed of a robot in moving between task locations	$sp_2 \in \{1, 2, 3, 4\}$ m/s
$r_{2,1}$	The rate for executing sub-task ($subT_1$)	0.0024
$r_{2,2}$	The rate for executing sub-task ($subT_2$)	0.0013
$r_{2,3}$	The rate for executing sub-task ($subT_3$)	0.001
$e_{2,1}$	The rate of energy consumption for executing sub-task ($subT_1$)	0.41 joules/second
$e_{2,2}$	The rate of energy consumption for executing sub-task ($subT_2$)	0.47 joules/second
$e_{2,3}$	The rate of energy consumption for executing sub-task ($subT_3$)	0.36 joules/second
$t_{2,1}$	The rate of time consumption for executing sub-task ($subT_1$)	0.012
$t_{2,2}$	The rate of time consumption for executing sub-task ($subT_2$)	0.017
$t_{2,3}$	The rate of time consumption for executing sub-task ($subT_3$)	0.006
Parameters of robot_3		
E_3	Amount of energy available per duty round	1100 Joules
B_3	Amount of total energy available for the mission purpose	5500 Joules
$sp3$	A variable for the speed of a robot in moving between task locations	$sp_3 \in \{1, 2, 3, 4\}$ m/s
$r_{3,1}$	The rate for executing sub-task ($subT_1$)	0.0024
$r_{3,2}$	The rate for executing sub-task ($subT_2$)	0.0013
$r_{3,3}$	The rate for executing sub-task ($subT_3$)	0.001
$e_{3,1}$	The rate of energy consumption for executing sub-task ($subT_1$)	0.39 joules/second
$e_{3,2}$	The rate of energy consumption for executing sub-task ($subT_2$)	0.45 joules/second
$e_{3,3}$	The rate of energy consumption for executing sub-task ($subT_3$)	0.34 joules/second
$t_{3,1}$	The rate of time consumption for executing sub-task ($subT_1$)	0.009
$t_{3,2}$	The rate of time consumption for executing sub-task ($subT_2$)	0.014
$t_{3,3}$	The rate of time consumption for executing sub-task ($subT_3$)	0.002

Table 5.8: Parameters defining the characteristics of robot i behaviour

  - $robot_2$ is assigned collection tasks (4, 5) and should consume energy and time less than (707.24, 706.6).
  - $robot_3$ is assigned collection tasks (6, 7, 8) and must consume energy and time less than (996.48, 1113.6).

- Between time instant t = 60s and t = 165s, the robots carry out their tasks as allocated, and regularly exchange heartbeats, run the local control loop.

- At time instant t = 166, two simultaneous disruptive events occurring at approximately the same time (i.e., the time between occurrence of the two events is approximately 6 seconds). In the first event, $robot_3$ experiences a slight increase in the probability p_{full_7}, which defines the likelihood that the robot would encounter a full recyclable bin. The local control loop of the robot manages to synthesise a local adaptation plan to restore its compliance with its CLA. The robot alter the configurable parameter ($p_{3,1}$), which defines the probability of collecting full recyclable bin (i.e., executing the sub-task ($subT_2$)). While in the second event, $robot_2$ undergoes a major local change, where the total energy available for the robot to execute its tasks has degraded by 25%. As a result, the robot can not achieve its CLA and initiate a new CLA re-negotiation, which leads to the following CLAs:
  - $robot_1$ is assigned collection tasks (1, 2, 3) and must consume energy and time less than (980.88, 1009.77).
  - $robot_2$ is assigned collection tasks (4) and should consume energy and time less than (447.14, 414.79).
  - $robot_3$ is assigned collection tasks (5, 6, 7) and must consume energy and time less than (996.48, 1113.6).

- Between time instant t = 234s and t = 920s, the team of robots carry out their tasks as allocated and continue to satisfy their CLAs.

- At time instant t = 921s, $robot_3$ experiences a complete failure that makes the robot unable to achieve its CLA. As a result, the robot stops sending its periodic heartbeats and loses contact with its peers ($robot_1$ and $robot_2$). After a short period of time (approximately 25 seconds), the nuDECIDE control instances running on $robot_1$ and $robot_2$ realise that $robot_3$ is missing and re-execute the partition of system goals control stage, which establishes the new CLAs as follow:
  - $robot_1$ is assigned collection tasks (1, 2, 3) and must consume energy and time less than (980.88, 1009.77).
  - $robot_2$ is assigned collection tasks (4) and should consume energy and time less than (447.14, 414.79).

- Between time instant t = 986s and t = 1416s, both robot$_1$ and robot$_2$ execute their tasks as allocated, and regularly the two robots exchange heartbeats.
- At time instant t = 1417s, robot$_1$ experiences a sudden increase in its workload, where the value of environment parameter (p_{full_3}) has increased by 80%. This parameter defines the probability, in which the robot may have to collect a full recyclable bin (i.e., the likelihood in which the robot may encounter the sub-task ($subT_2$)). As a result, the robot can not comply with its CLA and it notifies robot$_2$ with its updated contribution summary. Thus, the two robots identify the following new CLAs:
  - robot$_1$ is assigned collection tasks (1, 2) and must consume energy and time less than (649.84, 840.34).
  - robot$_2$ is assigned collection tasks (3) and should consume energy and time less than (447.14, 414.79).
- At approximately time instant t = 1933s, robot$_3$ rejoins the multi-robot system. The robot notifies its peers that it can contribute towards achieving system goals by computing and sharing its capability summary. Thus, the robot initiates a new CLA re-negotiation, which results in the following CLAs:
  - robot$_1$ is assigned collection tasks (1, 2) and must consume energy and time less than (649.84, 840.34).
  - robot$_2$ is assigned collection tasks (3) and should consume energy and time less than (447.14, 414.79).
  - robot$_3$ is assigned collection tasks (4, 5, 6) and must consume energy and time less than (996.48, 1113.6).

The results of the second type of experiments demonstrate the ability of the multi-robot system, which has been enhanced with the self-adaptive capabilities using nuDECIDE, to cope with various randomly generated disruptions. For example, the results indicate that the decentralised controllers are able to deal with disruptive events that may occur in two different robots, but simultaneously (e.g., as in time instant t = 166s), and the nuDECIDE control instance that coordinates the adaptation abilities of each robot can synthesise an adaptation plan as needed. More specifically, if the control instance can address the adaptation concerns through its local control loop by synthesising a local adaptation plan that satisfies its CLA, no notification is sent to its peers. If, however, the disruptive event can not be handled with local adaptation, the control instance recalculates its contribution summary, then its peers are notified, and the system goals are re-partitioned.

These observations contribute to building confidence that the approach works in general and can handle predefined and randomly generated disruptions, limiting its components' ability to

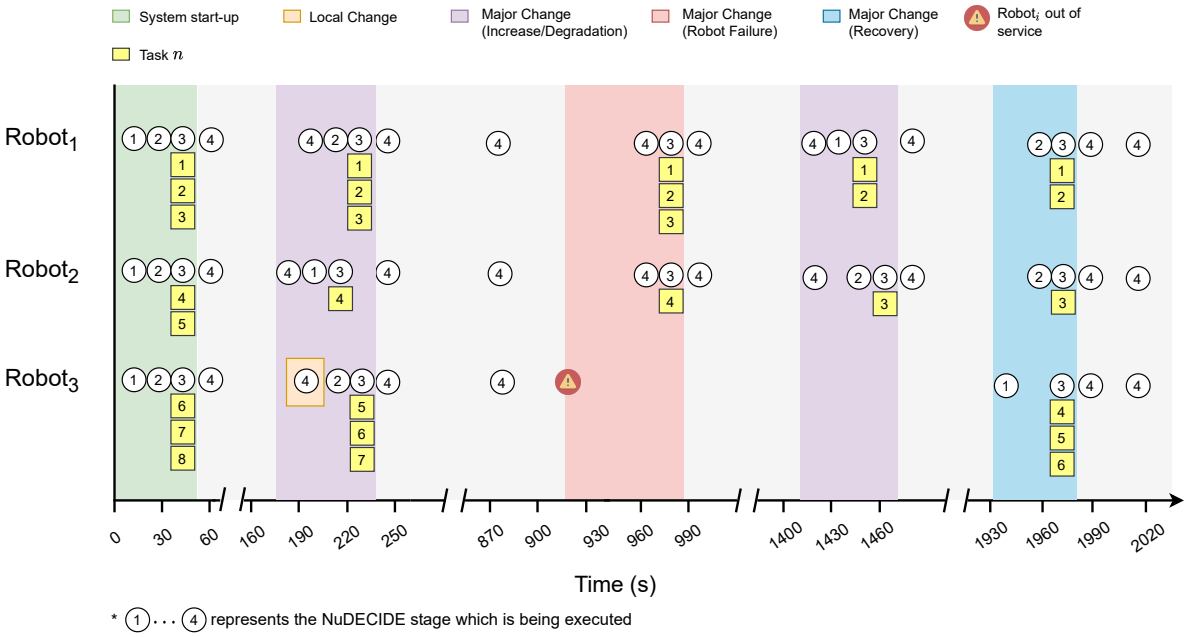

Figure 5.9: Execution of nuDECIDE stages 1–4 for a three-robot deployed in a mission involves random event

meet their local goals. In particular, the results from the two types of experiments show that the nuDECIDE variant using integer programming for the partition of the system goals can handle all types of events identified in Table 5.6, while the time required to respond to the different types of disruption varied between 5s and 60s.

5.2.4.2 Experiments for research question RQ2

For this research question, we executed the goal partitioning stage of the nuDECIDE decentralised control for robot teams comprising between $m = 2$ and $m = 66$ robots. The full simulation was not executed because all the other nuDECIDE stages only involve local computations with local models and local data. Therefore, assessing the scalability of nuDECIDE with integer programming only requires the evaluation of the goal partitioning stage, whose experimental results are shown in Figure 5.10, which we carried our for a problem instance with the same system parameters as in Tables 5.7 and 5.8. These experimental results, which were carried out with a 2.5 GHz Intel Core i7 MACBOOK Pro computer with 16GB memory, running Mac OSX 10.14., shown that time required for the goal partitioning grows approximately linearly with the number of robots, starting at under 0.02s for $m = 3$ robots and reaching above a fraction of second (0.1s) for the largest system we considered (with $m = 66$ robots). This shows that the approach is scalable to considerable robot team sizes.

Parameter	Description	Value
Parameters of robot_1		
E_1	Amount of energy available per duty round	1150 Joules
B_1	Amount of total energy available for the mission purpose	5750 Joules
sp_1	A variable for the speed of a robot in moving between task locations	$sp_1 \in$ {1, 2, 3, 4} m/s
$r_{1,1}$	The rate for executing sub-task ($subT_1$)	0.0024
$r_{1,2}$	The rate for executing sub-task ($subT_2$)	0.0013
$r_{1,3}$	The rate for executing sub-task ($subT_3$)	0.001
$e_{1,1}$	The rate of energy consumption for executing sub-task ($subT_1$)	0.41 joules/second
$e_{1,2}$	The rate of energy consumption for executing sub-task ($subT_2$)	0.47 joules/second
$e_{1,3}$	The rate of energy consumption for executing sub-task ($subT_3$)	0.36 joules/second
$t_{1,1}$	The rate of time consumption for executing sub-task ($subT_1$)	0.012
$t_{1,2}$	The rate of time consumption for executing sub-task ($subT_2$)	0.017
$t_{1,3}$	The rate of time consumption for executing sub-task ($subT_3$)	0.006
Parameters of robot$_2$		
E_2	Amount of energy available per duty round	1100 Joules
B_2	Amount of total energy available for the mission purpose	5500 Joules
sp_2	A variable for the speed of a robot in moving between task locations	$sp_2 \in$ {1, 2, 3, 4} m/s
$r_{2,1}$	The rate for executing sub-task ($subT_1$)	0.0034
$r_{2,2}$	The rate for executing sub-task ($subT_2$)	0.0019
$r_{2,3}$	The rate for executing sub-task ($subT_3$)	0.004
$e_{2,1}$	The rate of energy consumption for executing sub-task ($subT_1$)	0.46 joules/second
$e_{2,2}$	The rate of energy consumption for executing sub-task ($subT_2$)	0.51 joules/second
$e_{2,3}$	The rate of energy consumption for executing sub-task ($subT_3$)	0.39 joules/second
$t_{2,1}$	The rate of time consumption for executing sub-task ($subT_1$)	0.016
$t_{2,2}$	The rate of time consumption for executing sub-task ($subT_2$)	0.022
$t_{2,3}$	The rate of time consumption for executing sub-task ($subT_3$)	0.010
Parameters of robot_3		
E_3	Amount of energy available per duty round	1100 Joules
B_3	Amount of total energy available for the mission purpose	5500 Joules
$sp3$	A variable for the speed of a robot in moving between task locations	$sp_3 \in$ {1, 2, 3, 4} m/s
$r_{3,1}$	The rate for executing sub-task ($subT_1$)	0.0025
$r_{3,2}$	The rate for executing sub-task ($subT_2$)	0.0013
$r_{3,3}$	The rate for executing sub-task ($subT_3$)	0.001
$e_{3,1}$	The rate of energy consumption for executing sub-task ($subT_1$)	0.40 joules/second
$e_{3,2}$	The rate of energy consumption for executing sub-task ($subT_2$)	0.46 joules/second
$e_{3,3}$	The rate of energy consumption for executing sub-task ($subT_3$)	0.35 joules/second
$t_{3,1}$	The rate of time consumption for executing sub-task ($subT_1$)	0.011
$t_{3,2}$	The rate of time consumption for executing sub-task ($subT_2$)	0.016
$t_{3,3}$	The rate of time consumption for executing sub-task ($subT_3$)	0.007

Table 5.9: Parameters used for the second type of experiments that define the characteristics of the i-th behaviour

5.2.5 Discussion

The experiments we conducted in Section 5.2.4.1 demonstrate that nuDECIDE with the integer programming method can effectively support self-adaptation in a multi-robot system. The use of the integer programming method allows representing and partitioning system goals among the robots effectively. These experiments provide evidence of how effective the method is, whether the method was used at the start-up stage, where the team of robots is forming, and the system goals need to be allocated for the first time, or after the occurrence of disruptive events that require the re-allocation of system goals.

The results also demonstrate how the various control stages, from nuDECIDE control, behave in response to disruptions that affect a robot's ability to satisfy its CLA. For example, when the robot experiences an event that renders the robot unable to achieve its local goals, the local control loop of the affected robot tries to synthesise a local adaptation plan, which includes changing its configuration to restore the robot's compliance with its CLA. However, suppose the robot is unable to synthesise a local adaptation plan within the local control stage. In that case, it requests a global adaptation that would lead to selecting new CLAs for the multi-robot system.

Also, the experimental results illustrate that the estimated time for executing the control stages of nuDECIDE is relatively small, as the highest recorded time is 25 seconds, which was due to a total failure in one of the robots. In such an event, the remaining robot would detect that

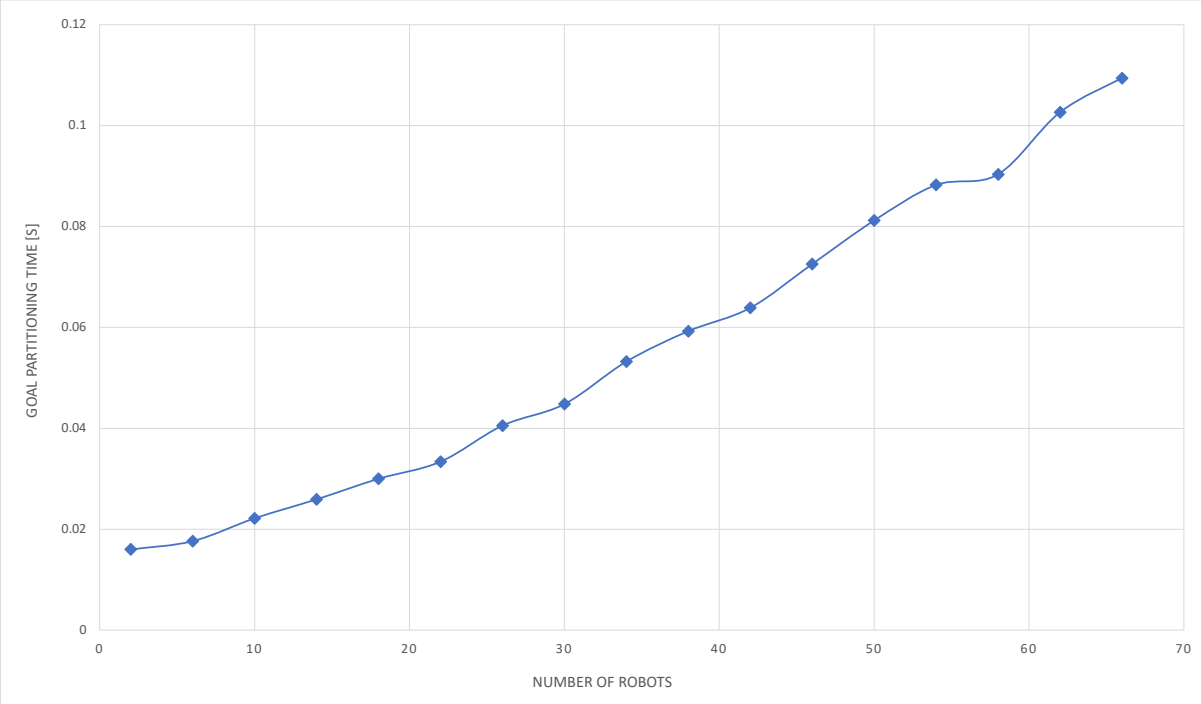

Figure 5.10: nuDECIDE scalability analysis for the use of integer programming method to partition system goals

a robot has failed after a short period of time. As for the experiments conducted in Section 5.2.4.2, the results show how scalable the approach is when the integer programming method is used to formulate and partition system goals. The results provide evidence on the impact of expanding the system's size by increasing the number of robots. The experiments simulate the execution of the goal partitioning stage using a variable size of robots. These experiments provide evidence that the estimated time for executing the goal partitioning in a system comprising $m = 66$ robots is relatively marginal. Indeed, especially when considering the size of the robots and the number of constraints included in the model. For instance, for a robot team comprising $m = 66$ robots, the number of decision variables and constraints included in the model that corresponds to this partitioning problem is 1056 variables and 662 constraints.

The waste management case study is selected as it provides insight into the scope of goal partitioning problems that can be represented using the integer program method. Specifically, the case study examines deploying a multi-robot system to a mission that includes a goal partitioning problem that can be formulated using the integer programming method. We believe that this case study includes requirements with similar characteristics to requirements for many applications [55, 56, 128]. By implementing and evaluating this case study, we aim at demonstrating that nuDECIDE can use the integer programming method to successfully adapt the decentralised robotic team.

This case study illustrates that nuDECIDE with the integer programming method can represent the behaviour characteristics of the robotic system along with the characteristics of the goal partitioning problem. Unlike the pipeline inspection case study, this case study uses continuous-time Markov chain (CTMC) models to captures the temporal characteristics of robots. For instance, CTMCs allow quantifying if the time required by a robot to finish its tasks is less than a threshold. Thus, the use of CTMCs rather than DTMCs in this case study complements the shortcomings in the pipeline inspection case study. As the waste management case study expands the characteristics of behaviour, which can be represented and reasoned about using nuDECIDE framework.

Also, this case study exhibits the use of the integer programming method to partition system goals. This method expands the characteristics of goal partitioning problems that can be represented using nuDECIDE. Unlike linear programming, integer programming includes decision variables that are restricted to be integer. In general integer programming is a discrete mathematical optimisation problem that comprises a linear objective function that includes integer decision variables. The optimisation of the objective function is subject to a set of linear constraints. Since the integer programming method uses discrete decision variables, many real-life problems can be formulated as integer programming problems. For example, assignment problems in which only a subset of the tasks can be undertaken at any one time. Another example is scheduling problems which necessitates respecting the precedence among tasks that need to be executed.

We believe that this case study provides insight into the scope of distributed systems that

can be enhanced with self-adaptation using nuDECIDE with integer programming method. In particular, the case study demonstrates the characteristics of goal partitioning problems that can be modelled using the integer programming method. Also, the case study illustrates the characteristics of requirements that can be reasoned about using nuDECIDE framework. The evaluation of this case study covers the efficacy and scalability aspects. It does not extend to consider the role of the confidence interval on optimising the system goals.

nuDECIDE is a pessimistic approach and relies on the use of conservative assumptions to configure how conservative is the contributions of a robot. This conservativeness leads to a less optimal configuration from a system point of view. These conservative assumptions allow a certain amount of "slack" for robots to handle environment uncertainties locally. While, they lead to an increase in the system-level costs (e.g., energy and time associated with achieving system goals). Further research is required to establish the tradeoff between the efficiency with which the robots operate, and the frequency with which the robots become unable to satisfy system goals.

5.3 Evaluation of MDP Policy Synthesis method

5.3.1 Research questions

In this section, we review several experiments aimed at evaluating the efficacy and scalability of nuDECIDE when using the MDP policy synthesis method for partitioning system goals. In particular, the use of the MDP policy synthesis method provides the ability to consider the uncertainty arising from the behaviour of some system components towards achieving its sub goals during the partition of system goals among its components. Considering this uncertainty in the behaviour of any of the system components towards achieving its objectives, helps to avoid repeated self-adaptation at the system-level. In general, this type of adaptation is more expensive than a component-level adaptation because it requires the concerted efforts of all components of the system to achieve it.

The use of the MDP policy synthesis method also provides the ability to represent the partition of system goals in the form of a probability distribution. This representation does not compromise with the required level of guarantees in achieving system goals. More specifically, the partition of system goals using this method provides the necessary guarantees for a system to achieve its goals whenever the system components adhere to their local goals. Below we review the research questions aimed at assessing the efficacy and scalability of the approach when using the MDP policy synthesis method in partitioning system goals.

RQ1 (Effectiveness) Can nuDECIDE with the MDP policy synthesis method handle different patterns of disruptions successfully, and what are the benefits in considering the probabilistic behaviour of the robots in the second variant of the method?

To address this research question, we performed several experiments aimed at evaluating the efficacy of nuDECIDE when using the MDP policy synthesis method for partitioning system goals. In particular, a number of experiments have been conducted that aim to compare the benefits of using an MDP model variant that represents the partition problem, considering the behaviour of a robot in implementing system goals, with another MDP model variant that represents the partition problem but does not factor the behaviour of a robot in achieving system goals.

RQ2 (scalability) How does the overhead of using nu-DECIDE with MDP policy synthesis method increase as the number of robots in the team grows?

We performed several experiments to assess the scalability of nuDECIDE when using MDP policy synthesis method to partition system goals. These experiments are conducted by simulating the steady increase in the number of robots from one experiment to another. The aim is to assess the impact of this increase on the timely synthesis of adaptation plans that partition the system goals. As in other experiments conducted to assess scalability, these experiments focus on the control phase that is concerned with the allocation of system goals because this control stage is the only control activity carried out at a system- level.

For evaluating the above research questions, we carried out several experiments using a 2.5 GHz Intel Core i7 MACBOOK Pro computer with 16GB memory, running Mac OSX 10.14. All these experiments include the execution of the goal partitioning stage of the nuDECIDE decentralised control for robot teams. In encoding the models used to represent the partition problem, we used PRISM's concrete syntax. While the partition of goals is carried out using the PRISM probabilistic model checking tool.

5.3.2 Case study: waste management

For the evaluation of MDP policy synthesis method, we used a slightly modified version of the waste management case study from Section 3.4.2. The change that we made to that case study involved extending models that describe the behaviour of the robots with a task utility reward structure that was used to quantify the rewards, which accumulate when a robot performs the task of collecting recyclable waste from full and non-full bins.

5.3.3 Simulation environment

We conducted the experiments for this part of the evaluation using the same simulation environment as described in Section 5.2.3.

5.3.4 Experimental results

5.3.4.1 RQ1 (Effectiveness)

For evaluation the effectiveness, we conducted two sets of experiments aimed at demonstrating how a robotic team that uses nuDECIDE could benefit from using the MDP policy synthesis method in the partition of system goals among its components. The two sets of experiments differ in terms of the characteristics of the MDP model used to allocate system goals. The first set of experiments was carried out using an MDP model variant that represents the partition problem and includes a probability distribution to capture a robot's behaviour towards achieving the system goals. While the second group of experiments was conducted using an MDP model variant that represents the partition problem but without considering the behaviour of robots towards achieving the system goals. For illustration purpose, we refer to the first set of experiments that uses an MDP model variant that includes the probability distribution as (UmP), while the second set of experiments that uses an MDP model variant without including such probability distribution is referred to as (UmWP).

Each set of experiments comprises 10 experiments, in which each experiment is performed using a simulated multi-robot mission whose characteristics and requirements are found in Section 3.5.2. Each mission simulates the deployment of ($m = 3$) robots to fulfil the mission requirements during a duty round. Each mission involves deploying the m-robot system to collect both garbage and recyclables from $n = 8$ physically distributed waste collections locations in a public park environment. In this section, we focus on describing the two sets of experiments that we ran in simulation in order to address the first research question (RQ1).

In general, the experiments carried out in each set involved creating a synthetic scenario in which two robots may experience disruptive events that affect their ability in collecting waste from the n-th location. More specifically, such disruptive events may occur from one experiment to another based on a probability distribution. The experiments from the two sets are similar in terms of the mission characteristics and requirements. These experiments are similar in terms of the probability distribution used to introduce disruptive events that may occur randomly from one experiment to another. In performing each set of experiments, we assumed that robot_2 and robot_3 could fail to carry out the waste collection task at location $n = 8$ during one-third of the experiments that were conducted within each set of experiments.

Regarding the first set of experiments, we assume that there is a probability distribution, which captures the behaviour of each robot toward carrying out tasks in locations $n = 1 \ldots 8$. This probability distribution represents the ability of a robot to execute tasks in a location. We further assume that robot_1 and robot_2 can execute tasks in all locations successfully with the probability 1. However, robot_3 is able to successfully perform the tasks in all locations, except for location ($n = 8$), as we assume that the robot may fail to reach the location to perform the tasks with the probability 0.33. We included this probability distribution in the MDP model that represents the partition problem and can capture robots' behaviour towards task execution.The evaluation of

Location n	Loc #1	Loc #2	Loc #3	Loc #4	Loc #5	Loc #6	Loc #7	Loc #8
Robot assignment	robot$_2$	robot$_2$	robot$_1$	robot$_2$	robot$_3$	robot$_1$	robot$_3$	robot$_3$

Table 5.10: Assignment of robot i to execute the tasks in the n-th location

the results and observations obtained from conducting the two sets of experiments was carried out in three stages. In the following, we describe the criteria that have been used to evaluate the results in each stage:

1. In the first stage (Stage 1), we use the result obtained from the first set of experiments, which are performed using an MDP model with probability (UmP), to compare the behaviour of each robot that is derived from simulating the robot execution of tasks during experiments, against the behaviour that is derived from the probabilistic model checking analysis that is carried out at the local control loop of each robot and involves analysing a CTMC model that describes the behaviour of a robot and its environment.

2. In the second stage (Stage 2), we use the result obtained from the second set of experiments, which are performed using an MDP model without probability (UmWP), to compare the behaviour of each robot that is derived from simulating the robot execution of tasks during experiments, against the behaviour that is derived from the probabilistic model checking analysis that is carried out at the local control loop of each robot and involves analysing a CTMC model that describes the behaviour of a robot and its environment.

3. Finally, in the last stage (Stage 3), we compare the results obtained from two previous stages (Stages 1 and 2) to demonstrate the benefit of using the MDP model variant from the first set of experiments against the alternative MDP model variant form the second set of experiments.

For illustration purpose, we refer to the behaviour that is derived from the simulation results as (SIM), while the behaviour that is derived from the model checking as (MC). As mentioned earlier in the evaluation criteria (List 5.3.4.1), our approach to evaluating the obtained results is based on comparing the behaviour of robots. More specifically, we use QoS metrics that are factored as reward structures in the CTMC model, which describes a robot's behaviour. These metrics are associated with a robot's behaviour towards achieving the tasks assigned to the robot during the partition of system goals. Namely, these metrics concern the utility, timeliness, and energy consumption, which can be used to quantify and compare a robots' task-execution behavior in line with all of the evaluation criteria. In the following, we describe the criteria that have been used to evaluate the results in each stage:

(Stage 1): evaluating results using an MDP model with probability (UmP) Table 5.10 shows the outcome of the partition of tasks among robots during the first set of experiments,

which were performed using the MDP model with the probability to represent the partition problem. As shown in the table, the tasks in location $n = 8$ were not assigned to $robot_3$ because the partition model considers the behaviour of robots in carrying out the tasks. Therefore, during the simulation of implementing the first set of experiments, none of the robots failed to perform the assigned tasks.

As Table 5.11 shows, the results obtained from the probabilistic model checking of the robots behaviour model are relatively close to the results obtained by simulating the robotics team's execution of tasks. In general, the results obtained using simulation for energy consumption, timeliness and utility metrics are better by 7.54%, 13.76%, and 1.39%, respectively, than those obtained using model checking. The multi-robot system consumed less energy and time when executing the tasks, and the system gained more utility due to robots completing more waste collection tasks.

Results obtained through model checking (MC) and using an MDP model with probability (UmP)			
Comparison criteria	Time	Energy	Utility
Total average	224.31	3569.72	12.03
Results obtained through simulation (SIM) and using an MDP model with probability (UmP)			
Comparison criteria	Time	Energy	Utility
Total average	207.39	3078.49	12.2
Delta	-16.92	-491.23	0.17

Table 5.11: Characteristics of the QoS metrics when using an mdp model *with* a probability distribution (UmP)

Stage 2: evaluating results using an MDP model without probability (UmWP): The results obtained from carrying out the second set of experiments are demonstrated in Table 5.13. Whereas, Table 5.12 shows the outcome of the partition of tasks among robots during the this set of experiments. The experiments comprise comparing the variation in the utility, energy consumed and timeliness metrics that are derived from the behaviour of robots obtained using model checking (MC) against the behaviour observed during the simulation of task execution by the multi-robot system. The table shows the discrepancy between the results obtained through

Location n	Loc #1	Loc #2	Loc #3	Loc #4	Loc #5	Loc #6	Loc #7	Loc #8
Robot assignment	robot$_2$	robot$_2$	robot$_1$	robot$_2$	robot$_3$	robot$_1$	robot$_3$	robot$_3$

Table 5.12: Table describing the assignment of robot i to execute the tasks in the n-th location

the probabilistic model checking of the robots behaviour model compared to the results obtained during the simulation. This variance is due to the failure of $robot_3$ in performing the tasks assigned to it in some experiments. In particular, the robot failed to perform the tasks in location $n = 8$, which affected the amount of utility that the multi-robot system obtained from carrying out the tasks. As shown in the table, the amount of utility had decreased by 8.9 % compared to the amount estimated during the probabilistic model checking stage. Also, the amount of energy and time consumed by the robotic team increased by 25 % and 13.86 %, respectively, due to the failure of $robot_3$ in achieving its local goals.

Results obtained through model checking (MC) and using an MDP model without probability (UmWP)			
Comparison criteria	Time	Energy	Utility
Total average	219.12	3524.37	11.87
Results obtained through simulation (SIM) and using an MDP model without probability (UmWP)			
Comparison criteria	Time	Energy	Utility
Total average	188.73	2639.972	10.802
Delta	-30.39	-884.398	-1.068

Table 5.13: Characteristics of the QoS metrics when using an mdp model *without* a probability distribution (UmWP)

5.3.4.2 RQ2 (Scalability)

To assess this research question, we conducted several experiments, which are limited to executing the goal partitioning stage of the nuDECIDE decentralised control. In particular, we conducted two sets of experiments; the first set comprises experiments that evaluate the scalability in the first variant of the MDP policy synthesis method that uses an MDP model that can represent robots' behaviour in implementing system goals. The second set of experiments assesses the other variant of the MDP policy synthesis method, which does not consider the robot's behaviour in partitioning system goals. In this section, we describe the experiment settings used in evaluating each variant of the method, along with the results obtained from carrying the experiments.

To evaluate the scalability in nuDECIDE with the first variant of the MDP policy synthesis method, we executed the goal partitioning stage of the nuDECIDE decentralised control for robot teams comprising between $m = 2$ and $m = 14$ robots. The plan was initially to simulate executing the partition stage using teams of robots ranging in size from 2 to 32. Still, the results from Figure 5.11 shows the time required for the goal partitioning grows exponentially with the number of robots, starting at under fraction of a second (0.014s) for $m = 2$ robots, 0.60 second for $m = 6$, and reaching 19.6 seconds for the largest system we considered (with $m = 14$ robots).

As for the second set of experiments, we simulated the execution of the goal partitioning stage for robot teams comprising between $m = 2$ and $m = 26$ robots. Figure 5.12 illustrate the results obtained from carrying out these experiments, which aim to evaluate the scalability in nuDECIDE with the second variant of the MDP policy synthesis method. As with the case of the first variant of the method, the time required grows exponentially with the number of robots, starting at under a fraction of a second (0.01s) for $m = 2$ robots, 0.07 second for $m = 6$, and reaching 9.86 seconds for the largest system we considered (with $m = 26$ robots).

As shown in Figure 5.11 and 5.12, the time required to partition system goals using the two variants of the method grows exponentially with the size of the multi-robot system (with slightly smaller times for the second variant of the MDP policy synthesis method compared to the first variant). This is expected for a model checking technique, and means that this goal partitioning method is only applicable to systems with a relatively small number of components.

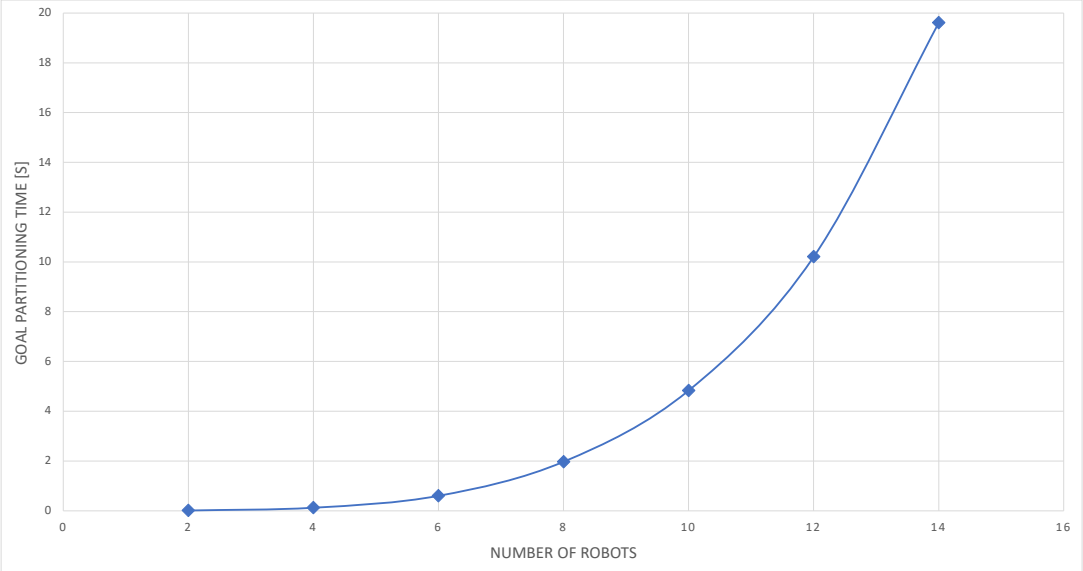

Figure 5.11: nuDECIDE scalability analysis for the use of the first variant of MDP policy synthesis method to partition system goals

5.3.5 Discussion

The experiments in Section 5.3.4.1, were conducted to evaluate the effectiveness of nuDECIDE with two variants of the MDP policy synthesis method. The results illustrate that the method variant that considers robots' behaviour when system goals are partitioned among robots outperform results obtained from simulating the use of the alternative variant of this method. More specifically, the results obtained through simulating the execution of tasks by the multi-robot are relatively consistent with the system's behaviour derived from the probabilistic model checking. The results in metrics such as utility, energy consumption, and timeliness associated with task

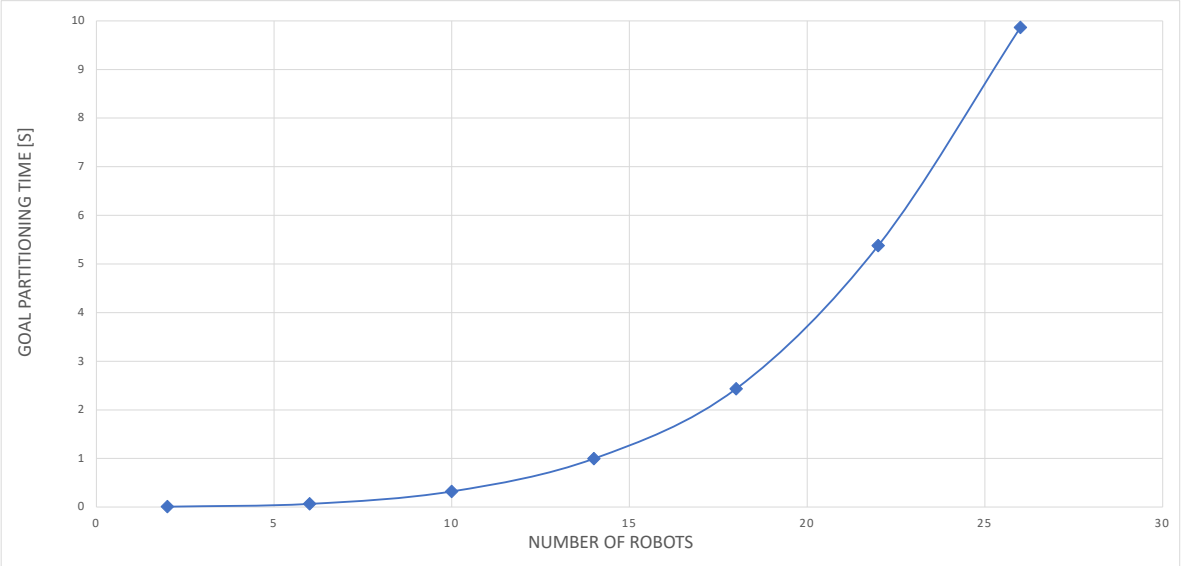

Figure 5.12: nuDECIDE scalability analysis for the use of the second variant of MDP policy synthesis method to partition system goals

execution indicate that the behaviour obtained from simulation outperform the one derived from model checking. On the other hand, the results obtained through using the second variant of this method indicate that the behaviour derived from model checking the task execution is relatively better than what obtained from simulating the task execution by the team of robots. Thus, considering robots' behaviour in partitioning system goals can add to the effectiveness and efficiency of a nuDECIDE system comprising a team of robots.

The results demonstrated in Section 5.3.4.2, which are obtained from simulating the execution of the goal partitioning stage using the two variants of the MDP policy synthesis method, indicate the limitation in applying these methods in robot teams comprising a relatively large number of robots. The results illustrate the exponential growth pattern of the required time when the system's size increases.

This case study illustrates the differences when using nuDECIDE with the MDP strategy synthesis method instead of the integer programming method to partition system goals. In particular, the case study is chosen to highlight the differences between the two methods and to show the characteristics of goal partitioning problems that can be supported by each method. In contrast to the integer programming method, the MDP strategy synthesis method can account for uncertainty in allocating system goals. In general, the latter method allows capturing the behaviour of robots towards achieving system goals. Moreover, the two methods differ in terms of the outcome of the goal partition, when using the MDP strategy synthesis method, the outcome is a strategy that allocates the goals of the system using probability distributions. While in the case of using the integer programming method, the outcome of the partition assigns each sub-goal of the system to a specific robot in a deterministic manner.

Similar to the integer programming method, we believe that the MDP strategy synthesis method can be applied to partition problems that are characterised as a pure discrete optimisation problem [102] where all of the decision variables used are restricted to be an integer. However, further research is required to assess the applicability of these methods to partition problems from additional domains. Also, all the goal partition problems that are considered in the context of this thesis consider a single optimisation metric (i,e. namely optimising the covered distance, consumed energy and gained utility when using the linear program, integer program and MDP strategy synthesis methods, respectively). Further research is required to expand the applicability of nuDECIDE to domains where the partition problem includes several optimisation metrics that need to be optimised simultaneously.

This case study demonstrates the characteristics of goal partition problems that can be represented using nuDECIDE with the MDP policy synthesis method. We believe that the findings of the case study show the advantages of using the MDP model variant that factors for the uncertainty in achieving system goals compared to the model variant that does not account for such uncertainty. Also, the findings demonstrate the drawbacks in regard to the scalability concerns when using the MDP strategy synthesis method compared with the use of the integer programming method.

5.4 Threats to Validity

This section reviews many of the factors that affect the validity of the findings presented in this chapter. These factors are classified according to the origin of the threats. In general, validity threats arise from internal and external factors. Below, we review and discuss the various factors that affect the validity of the findings presented in this chapter.

External validity factors may pose threats to the applicability of the nuDECIDE framework to developing the decentralised control software for a wide range of systems. Such threats are evident when the characteristics of the multi-robot system and its requirements are not captured in the three case studies (i.e., introduced in Chapter 3), which illustrate the effectiveness of the nuDECIDE framework for implementing several distributed SAS from the CPS domain. To expand the range of systems that can benefit from nuDECIDE approach, we introduced three alternative methods to partition distributed SAS goals based on the contributions that SAS components can make toward achieving these goals. Each method represents a particular class of system goals, demonstrating that the various goal partitioning formalisms defined in Chapter 3 are sufficient to expand the applicability of the nuDECIDE framework to a wide range of applications.

From reviewing the literature, numerous realistic robotic team missions exist whose requirements can be formulated using linear programming [48, 190], integer programming [5, 7, 58]

and MDP policy synthesis [67, 158, 166] methods. Although we believe that the nuDECIDE framework can support the development of decentralised control software for distributed SAS systems in other domains, the variance in the characteristics of such systems and their requirements may pose threats to validity. Further evaluation of the applicability of the nuDECIDE framework in applications from different domains such as service-based [35], IoT systems [137]), etc. is required to assess the generalisability of our approach. Also, further evaluation is required to assert that nuDECIDE framework is able to accommodate employing more mathematical programming methods to expand the applicability of the approach to a new domain. Methods such as network flow optimisation [185], constraint programming [176], nonlinear programming [19], and multiple criteria optimisation could enable the development of distributed SASs not supported by the techniques covered in this thesis.

Another external threat factor may arise if models required to partition the system goals or analyse the behaviour of a nuDECIDE component are too large for addressing adaptation concerns in a timely manner. This threat can be alleviated by extending the nuDECIDE framework to support hierarchical goal decomposition. The hierarchical decomposition of system goals enables forming different component teams or groups. The various teams share a common objective on a system level but differ in terms of the roles in which each team has to carry to satisfy the common objective. The teams can be formed based on collaboration rules as in [94], which enable dividing system goals into sub-goals captured using relatively smaller models that can be analysed efficiently.

Internal threats to validity may stem from the methods used to perform experiments and analyse the results. In particular, threats to validity may arise from the use of simulation in conducting the experiments for evaluating the effectiveness and scalability of distributed SASs that are developed using nuDECIDE. Such threats are mitigated by simulating the execution of missions that include a wide range of scenarios with different characteristics in terms of the number of robots and the mechanism in which failure/degradation patterns are introduced during the simulation. Also, the results were obtained from several independent experiments. The real mobile robots were substituted with deploying software components that simulate the use of robots in conducting experiments. Each software component communicates with its nuDECIDE control instance to send heartbeat messages and environment readings, and receive configurable parameters that determine its behaviour.

According to Banks [14], the use of simulation in experimental studies is an established approach for evaluating research in software engineering. The simulation involves imitating the operation of a system over time and aims to create an artificial history that includes the operating characteristics of the system being represented. The artificial history is subsequently used to infer conclusions about the observed system. In general, simulation is used to reduce the time, costs and risks of experiments and also facilitate the reproducibility of experimental studies as simulations are usually conducted in virtual environments. In the context of this

thesis, simulating the use of robots has contributed to reducing the time and costs of conducting experiments that examine the effectiveness and scalability of nuDECIDE when using the various goal partition methods. In particular, simulating the conduct of these experiments made it possible to anticipate the effects of implementing such experiments. We believe that relying on simulation in conducting those experiments facilitates the reproducibility of the generated results and contributes to raising the productivity of the research and the scope of the experiments.

5.5 Summary

This chapter presented three case studies used to evaluate the nuDECIDE approach, including its different system-goal partitioning techniques and reusable software platform. The first case study used a team of three mobile robots that emulated a pipeline inspection mission in a lab testbed at the University of York, and employed linear programming to partition the pipeline route requiring inspection among these robots, which were assumed to have different types of sensors. The second case study involved the simulation of a waste-management autonomous system in which a team of mobile robots collected both garbage and recyclables from multiple locations within a public park. Given the fact that each robot could only be allocated an integer number of such locations in this case study, the goal partitioning for the system was encoded as an integer programming problem and solved accordingly. Finally, the third case study also simulates the execution of a waste management mission using a team of mobile robots. however, this case study includes the use of two variants of the MDP policy synthesis method in partitioning system goals. Unlike the linear and integer programming methods, the MDP method can consider the behavior of system components when partitioning system goals. However, this advantage of using MDP policy synthesis comes with a significant increase in computational overheads, which grow exponentially with the number of system components, limiting the applicability of the method to small and medium sized systems.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusions

There is a growing interest in approaches that support self-adaptation in distributed systems. In particular, the focus is on engineering approaches that relies on decentralised software controllers to address self-adaptation concerns in business-critical applications [33]. For instance, many applications from the CPS and IoT domains often deal with strict dependability and performance requirements. Such systems must adapt to dynamic changes that occur as a result of interacting with their physical environment. These systems' requirements often necessitate that the adopted self-adaptive approach should provide guarantees in which the system complies with its strict requirements.

This thesis provided several contributions to the engineering of decentralised control software for such systems. First, a repertoire of formal methods for partitioning the goals of a distributed SAS among its components was introduced. These methods make novel use of three mathematical programming techniques to enable the engineering of decentralised-control distributed SAS with different types of goals. A goal-partitioning method based on linear programming enables the development of such system whose goals can be specified in terms of linear constraints and a linear optimisation objective at system level. Another goal-partitioning method, based on integer programming whose goals can be represented and solved using integer programming formulations. The partition model of such method includes discrete decision variables that are used to formulate non-linear equations and inequalities. Finally, a goal-partitioning method employing MDP policy synthesis supports employing models capable of representing the problem of partitioning system goals and account for the possibility that system components may fail in achieving their contributions towards system goals. In particular, the method is feasible

for applications whose non-determinism cannot be easily expressed as an linear or integer programming problem. We introduced two variants of the MDP policy synthesis method. The first variant of the method can account for the uncertainty in the behavior of system components when system goals are partitioned. While, the second variant does not consider such uncertainty in components' behaviour, however the synthesised partition in both variants can include probability distributions that describe the assignment of goals to system components.

A second major contribution of the thesis is the development of an application-independent software architecture and a reusable software platform for decentralised control software for distributed SAS. These are complemented by an engineering approach for specialising the platform to enable the development of a distributed SAS for a given application.

All contributions summarised above were evaluated through three case studies involving distributed SAS comprising multiple mobile robots. The first of these case studies used a team of real mobile robots in a lab testbed, and confirmed the applicability of linear-programming goal partitioning. The second case study involves simulating the execution of a waste management mission using a team of mobile robots. The mission comprises deploying these robot to collect both garbage and recyclables from multiple locations with a public park. The system goals in this case study are encoded as as an integer programming problem and solved accordingly. The last case study also simulates the execution of a waste management mission using a team of mobile robots. however, this case study includes the use of two variants of the MDP policy synthesis method in partitioning system goals.

While our experimental results show how the research contributions provided in the thesis advance the engineering of distributed SAS with decentralised control, they also identify several limitations of the new results. In particular, the goal partitioning techniques introduced in Chapter 3 require non-trivial execution time (sub-second for linear programming, but up to tens of seconds for integer programming). As such, these techniques can only be used for distributed SAS where the major changes triggering new executions of the goal partitioning are infrequent, e.g., not happening more frequently than once every few tens of minutes.

The second limitation stems from employing a fully decentralised system-level MAPE loop architecture. In such architecture nuDECIDE component has to perform the system-level goal partitioning stage redundantly. In comparison, a fully decentralised system-level MAPE loop has significant benefits, such as eliminating a single point of failure and reducing the need for inter-component communication and synchronisation [63]. Alternative variants of the architecture can use other decentralised MAPE loop patterns [184]. In particular, the centralised execution of the Planning has the advantage that each component does not redundantly perform the system-level goal partitioning, and therefore it can employ techniques that may not yield the same partition each time they are applied (e.g., metaheuristics as in [83]).

Since nuDECIDE employs a fully decentralised system-level MAPE loop architecture, the partitioning of system goals is carried out at the system-level. Such architecture prioritises

component goals over system ones, which leads to the synthesis of a sub-optimal adaptation plan from a system viewpoint. As the synthesis of such plans is performed under conservative assumptions, in which a system component can achieve more towards achieving the system objective, but the component report that it can achieve less (i.e., the local capability analysis control stage computes the contribution summary under conservative assumptions).

Finally, one further limitation is due to the evaluation of our research contributions exclusively for multi-robot SAS. While we expect the results to be equally applicable to other distributed component-based systems (e.g., multi-agent software and service-based systems), this generality of the research remains a hypothesis.

6.2 Future work

Many extensions of the research contributions presented in the thesis are possible. This section provides a summary of some of the main future work directions that would expand the applicability of our nuDECIDE research.

First, the adaptation of additional mathematical programming techniques for use within our nuDECIDE framework could enable the development of distributed self-adaptive systems not supported by the techniques covered in the thesis. Such additional technique could include mixed-integer programming, network flow and constraint programming [24, 185].

Second, the project described in the thesis evaluated the nuDECIDE framework exclusively for multi-robot system. To demonstrate the generality of the framework, further work may include its specialisation for other types of distributed self-adaptive systems, such as IoT systems [97] and multi-agent software systems.

Third, the current nuDECIDE partition methods do not support defining any dependencies between tasks, e.g., executing a given task may only be possible after another task was executed, or a given task cannot be assigned to a system component unless another task is assigned to the same component. In the current nuDECIDE framework, any such dependencies have to be considered and encoded manually when the system goals are partitioned. Extending nuDECIDE with a domain-specific language for specifying such task dependencies and with a constraint solver such as Alloy [99] for handling these dependencies along the lines of the research from [40–42] would greatly expand the types of distributed self-adaptive systems supported by the framework.

Finally, the nuDECIDE framework does not currently support hierarchical goal decomposition, so that different component teams or groups can satisfy goals that contribute to others that exist at a higher level in the hierarchy. Extending nuDECIDE with the capability of forming teams based on collaboration rules will enable the framework to support larger systems and higher degrees of flexibility in the way in which the goals are partitioned and satisfied.

BIBLIOGRAPHY

[1] "An architectural blueprint for autonomic computing," IBM Corporation, Tech. Rep., 06 2006.

[2] J. Aldrich, D. Garlan, C. Kästner, C. Le Goues, A. Mohseni-Kabir, I. Ruchkin, S. Samuel, B. Schmerl, C. S. Timperley, M. Veloso *et al.*, "Model-based adaptation for robotics software," *IEEE Software*, vol. 36, no. 2, pp. 83–90, 2019.

[3] J. Andersson, R. De Lemos, S. Malek, and D. Weyns, "Reflecting on self-adaptive software systems," in *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on*. IEEE, 2009, pp. 38–47.

[4] R. C. Arkin, *Behavior-based robotics*. MIT press, 1998.

[5] B. A. Asfora, J. Banfi, and M. Campbell, "Mixed-integer linear programming models for multi-robot non-adversarial search," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6805–6812, 2020.

[6] K. Ashton, "That 'internet of things' thing," *RFiD Journal*, vol. 22, no. 7, pp. 97–114, 2009.

[7] N. Atay and B. Bayazit, "Mixed-integer linear programming solution to multi-robot task allocation problem. washington univ. st," Louis, Tech. Rep, Tech. Rep., 2006.

[8] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying continuous time markov chains," in *International Conference on Computer Aided Verification*. Springer, 1996, pp. 269–276.

[9] ——, "Model-checking continuous-time markov chains," *ACM Transactions on Computational Logic (TOCL)*, vol. 1, no. 1, pp. 162–170, 2000.

[10] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-checking algorithms for continuous-time markov chains," *IEEE Transactions on software engineering*, vol. 29, no. 6, pp. 524–541, 2003.

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

[12] C. Baier, J.-P. Katoen, and H. Hermanns, "Approximative symbolic model checking of continuous-time markov chains," in *International Conference on Concurrency Theory*. Springer, 1999, pp. 146–161.

[13] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 295–310, 2004.

[14] J. Banks, "Introduction to simulation," in *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*, 1999, pp. 7–13.

[15] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini, "Validation of web service compositions," *IET software*, vol. 1, no. 6, pp. 219–232, 2007.

[16] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[17] A. Bennaceur, C. Ghezzi, K. Tei, T. Kehrer, D. Weyns, R. Calinescu, S. Dustdar, Z. Hu, S. Honiden, F. Ishikawa *et al.*, "Modelling and analysing resilient cyber-physical systems," in *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2019, pp. 70–76.

[18] J. O. Berger, *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

[19] D. P. Bertsekas, "Nonlinear programming," *Journal of the Operational Research Society*, vol. 48, no. 3, pp. 334–334, 1997.

[20] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*. Athena Scientific Belmont, MA, 1997, vol. 6.

[21] A. Bianco and L. De Alfaro, "Model checking of probabilistic and nondeterministic systems," in *Foundations of Software Technology and Theoretical Computer Science*. Springer, 1995, pp. 499–513.

[22] A. Borda, L. Pasquale, V. Koutavas, and B. Nuseibeh, "Compositional verification of self-adaptive cyber-physical systems," in *2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2018, pp. 1–11.

[23] J. S. Bradbury, J. R. Cordy, J. Dingel, and M. Wermelinger, "A survey of self-management in dynamic software architecture specifications," in *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*. ACM, 2004, pp. 28–33.

[24] S. P. Bradley, A. C. Hax, and T. L. Magnanti, "Applied mathematical programming," 1977.

[25] Y. Brun, "Efficient 3-sat algorithms in the tile assembly model," *Natural Computing*, vol. 11, no. 2, pp. 209–229, 2012.

[26] Y. Brun, R. Desmarais, K. Geihs, M. Litoiu, A. Lopes, M. Shaw, and M. Smit, "A design space for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 33–50.

[27] Y. Brun and N. Medvidovic, "An architectural style for solving computationally intensive problems on large networks," in *Software Engineering for Adaptive and Self-Managing Systems, 2007. ICSE Workshops SEAMS'07. International Workshop on*. IEEE, 2007, pp. 2–2.

[28] Y. Brun and D. Reishus, "Path finding in the tile assembly model," *Theoretical Computer Science*, vol. 410, no. 15, pp. 1461–1472, 2009.

[29] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Engineering self-adaptive systems through feedback loops," in *Software engineering for self-adaptive systems*. Springer, 2009, pp. 48–70.

[30] T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil, "Deeco: an ensemble-based component system," in *Proceedings of the 16th International ACM Sigsoft symposium on Component-based software engineering*. ACM, 2013, pp. 81–90.

[31] R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar, I. Habli, and T. Kelly, "Engineering trustworthy self-adaptive software with dynamic assurance cases," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2017.

[32] R. Calinescu, "General-purpose autonomic computing," in *Autonomic Computing and Networking*. Springer, 2009, pp. 3–30.

[33] R. Calinescu, J. Cámara, and C. Paterson, "Socio-cyber-physical systems: models, opportunities, open challenges," in *2019 IEEE/ACM 5th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*. IEEE, 2019, pp. 2–6.

[34] R. Calinescu, S. Gerasimou, and A. Banks, "Self-adaptive software with decentralised control loops," in *International Conference on Fundamental Approaches To Software Engineering*. Springer, 2015, pp. 235–251.

[35] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Communications of the ACM*, vol. 55, no. 9, pp. 69–77, 2012.

[36] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 387–409, 2010.

[37] ——, "Dynamic qos management and optimization in service-based systems," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 387–409, 2011.

[38] R. Calinescu and S. Kikuchi, "Formal methods@ runtime," in *Monterey Workshop*. Springer, 2010, pp. 122–135.

[39] R. Calinescu and M. Kwiatkowska, "Using quantitative analysis to implement autonomic it systems," in *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. IEEE, 2009, pp. 100–110.

[40] J. Cámara, "Haiq: Synthesis of software design spaces with structural and probabilistic guarantees," in *Proceedings of the 8th International Conference on Formal Methods in Software Engineering*, 2020, pp. 22–33.

[41] J. Cámara, D. Garlan, and B. Schmerl, "Synthesizing tradeoff spaces with quantitative guarantees for families of software systems," *Journal of Systems and Software*, vol. 152, pp. 33–49, 2019.

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

[43] G. Casella and R. L. Berger, *Statistical inference*. Duxbury Pacific Grove, CA, 2002, vol. 2.

[44] P. Castillejo, B. Cürüklü, R. Fresco, G. Johansen, S. Bilbao-Arechabala, B. Martínez-Rodríguez, L. Pomante, J.-F. Martínez-Ortega, and M. Santic, "The afarcloud ecsel project," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 414–419.

[45] A. Charnes and W. W. Cooper, "Management models and industrial applications of linear programming," *Management Science*, vol. 4, no. 1, pp. 38–91, 1957.

[46] B. H. Cheng, R. de Lemos, D. Garlan, H. Giese, M. Litoiu, J. Magee, H. A. Muller, and R. Taylor, "Seams 2009: Software engineering for adaptive and self-managing systems," in *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*. IEEE, 2009, pp. 463–464.

[47] B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee *et al.*, "Software engineering for self-adaptive systems: A research roadmap," in *Software engineering for self-adaptive systems*. Springer, 2009, pp. 1–26.

[48] F.-T. Cheng and D. E. Orin, "Efficient algorithm for optimal force distribution-the compact-dual lp method," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 2, pp. 178–187, 1990.

[49] S.-W. Cheng, D. Garlan, and B. Schmerl, "Making self-adaptation an engineering reality," in *Self-star properties in complex information systems*. Springer, 2005, pp. 158–173.

[50] ——, "Evaluating the effectiveness of the rainbow self-adaptive system," in *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 2009, pp. 132–141.

[51] D. Claes, F. Oliehoek, H. Baier, K. Tuyls *et al.*, "Decentralised online planning for multi-robot warehouse commissioning," in *AAMAS'17: Proceedings of the 16th International Conference on Autonomous Agents and Multi Agent Systems*, 2017, pp. 492–500.

[52] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 8, no. 2, pp. 244–263, 1986.

[53] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, "Model checking and the state explosion problem," in *LASER Summer School on Software Engineering*. Springer, 2011, pp. 1–30.

[54] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys (CSUR)*, vol. 28, no. 4, pp. 626–643, 1996.

[55] B. Coltin, M. Veloso, and R. Ventura, "Dynamic user task scheduling for mobile robots," in *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[56] M. Conforti, G. Cornuéjols, and G. Zambelli, *Integer programming*. Springer, 2014, vol. 271.

[57] G. Dantzig, *Linear programming and extensions*. Princeton university press, 2016.

[58] M. Darrah, W. Niland, and B. Stolarik, "Multiple uav dynamic task allocation using mixed integer linear programming in a sead mission," in *Infotech@ Aerospace*, 2005, p. 7164.

[59] R. Das, J. O. Kephart, C. Lefurgy, G. Tesauro, D. W. Levine, and H. Chan, "Autonomic multi-agent management of power and performance in data centers," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*. Citeseer, 2008, pp. 107–114.

[60] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*. McGraw-Hill Higher Education New York, 2008.

[61] C. Daws, "Symbolic and parametric model checking of discrete-time markov chains," in *International Colloquium on Theoretical Aspects of Computing*. Springer, 2004, pp. 280–294.

[62] R. De Lemos and J. L. Fiadeiro, "An architectural support for self-adaptive software for treating faults," in *Proceedings of the first workshop on Self-healing systems*. ACM, 2002, pp. 39–42.

[63] R. de Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson *et al.*, "Software engineering for self-adaptive systems: research challenges in the provision of assurances," in *Software Engineering for Self-Adaptive Systems III. Assurances*. Springer, 2017, pp. 3–30.

[64] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 1–32.

[65] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A storm is coming: A modern probabilistic model checker," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 592–600.

[66] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester, *A Modern Introduction to Probability and Statistics: Understanding why and how*. Springer Science & Business Media, 2005.

[67] U. M. Diwekar, "Optimization under uncertainty," in *Introduction to Applied Optimization*. Springer, 2020, pp. 151–215.

[68] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A survey of autonomic communications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 2, pp. 223–259, 2006.

[69] J. Dowling and V. Cahill, "The k-component architecture meta-model for self-adaptive software," in *International Conference on Metalevel Architectures and Reflection*. Springer, 2001, pp. 81–88.

[70] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 2009, pp. 111–121.

[71] D. Evans, "Top 25 technology predictions," *Cisco IBSG Innovations Practice*, 2009.

[72] J. Ferber, *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley Reading, 1999, vol. 1.

[73] A. Filieri, C. Ghezzi, and G. Tamburrelli, "A formal approach to adaptive software: continuous assurance of non-functional requirements," *Formal Aspects of Computing*, vol. 24, no. 2, pp. 163–186, 2012.

[74] B. Fitzgerald, "Software crisis 2.0," *Computer*, vol. 45, no. 4, pp. 89–91, 2012.

[75] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven, "Using architecture models for runtime adaptability," *IEEE software*, vol. 23, no. 2, pp. 62–70, 2006.

[76] I. O. for Standardization and I. E. Commission, *Software Engineering–Product Quality: Quality model*. ISO/IEC, 2001, vol. 1.

[77] A. Gambi, M. Pezze, and M. Young, "Sla protection models for virtualized data centers," in *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on*. IEEE, 2009, pp. 10–19.

[78] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. Le Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig, "The SCIP Optimization Suite 7.0," Optimization Online, Technical Report, March 2020. [Online]. Available: http://www.optimization-online.org/DB_HTML/2020/03/7705.html

[79] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[80] S. I. Gass, *Linear programming: methods and applications*. Courier Corporation, 2003.

[81] K. Genova and V. Guliashki, "Linear integer programming methods and approaches–a survey," *Journal of Cybernetics and Information Technologies*, vol. 11, no. 1, 2011.

[82] I. Georgiadis, J. Magee, and J. Kramer, "Self-organising software architectures for distributed systems," in *Proceedings of the first workshop on Self-healing systems*. ACM, 2002, pp. 33–38.

[83] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for quality-of-service software engineering (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 319–330.

[84] C. Ghezzi, "Evolution, adaptation, and the quest for incrementality," in *Monterey Workshop*. Springer, 2012, pp. 369–379.

[85] C. Ghezzi, M. Pezzè, M. Sama, and G. Tamburrelli, "Mining behavior models from user-intensive web applications," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 277–287.

[86] C. Ghezzi and A. M. Sharifloo, "Dealing with non-functional requirements for adaptive systems via dynamic software product-lines," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 191–213.

[87] ——, "Model-based verification of quantitative non-functional properties for software product lines," *Information and Software Technology*, vol. 55, no. 3, pp. 508–524, 2013.

[88] C. Ghezzi and G. Tamburrelli, "Reasoning on non-functional requirements for integrated services," in *Requirements Engineering Conference, 2009. RE'09. 17th IEEE International*. IEEE, 2009, pp. 69–78.

[89] S. Gokhale and K. S. Trivedi, "Structure-based software reliability prediction," in *Proc. of Fifth Intl. Conference on Advanced Computing*, 1997.

[90] M. Handte, G. Schiele, V. Matjuntke, C. Becker, and P. J. Marrón, "3pc: System support for adaptive peer-to-peer pervasive computing," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 7, no. 1, p. 10, 2012.

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

[92] M. G. Hinchey and R. Sterritt, "Self-managing software," *Computer*, vol. 39, no. 2, pp. 107–109, 2006.

[93] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "Prism: A tool for automatic verification of probabilistic systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2006, pp. 441–444.

[94] P. Hnetynka, T. Bures, I. Gerostathopoulos, and J. Pacovsky, "Using component ensembles for modeling autonomic component collaboration in smart farming," in *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2020, pp. 156–162.

[95] P. Horn, "Autonomic computing: Ibm's perspective on the state of information technology," 2001.

[96] M. U. Iftikhar and D. Weyns, "Activforms: Active formal models for self-adaptation," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2014, pp. 125–134.

[97] M. U. Iftikhar, G. S. Ramachandran, P. Bollansée, D. Weyns, and D. Hughes, "DeltaIoT: a self-adaptive internet of things exemplar," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2017 IEEE/ACM 12th International Symposium on*. IEEE, 2017, pp. 76–82.

[98] P. Inc. (2017) Activitybot overview. [Online]. Available: https://www.parallax.com/robots/robots-overview

[99] D. Jackson, "Alloy: a lightweight object modelling notation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 2, pp. 256–290, 2002.

[100] M. Jackson, "The meaning of requirements," *Ann. Softw. Eng.*, vol. 3, pp. 5–21, Jan. 1997. [Online]. Available: http://dl.acm.org/citation.cfm?id=590564.590577

[101] P. Kall, S. W. Wallace, and P. Kall, *Stochastic programming*. Springer, 1994.

[102] J. Kallrath and A. Schreieck, "Discrete optimisation and real world problems," in *International Conference on High-Performance Computing and Networking*. Springer, 1995, pp. 351–359.

[103] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, 1984, pp. 302–311.

[104] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen, "The ins and outs of the probabilistic model checker mrmc," *Performance evaluation*, vol. 68, no. 2, pp. 90–104, 2011.

[105] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[106] C. A. Kerrache, N. Lagraa, C. T. Calafate, and A. Lakas, "Tfdd: A trust-based framework for reliable data delivery and dos defense in vanets," *Vehicular Communications*, vol. 9, pp. 254–267, 2017.

[107] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," *Cooperative Robots and Sensor Networks 2015*, pp. 31–51, 2015.

[108] M. Kim, M.-O. Stehr, J. Kim, and S. Ha, "An application framework for loosely coupled networked cyber-physical systems," in *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. IEEE, 2010, pp. 144–153.

[109] V. Klee and G. J. Minty, "How good is the simplex algorithm," *Inequalities*, vol. 3, no. 3, pp. 159–175, 1972.

[110] D. E. Knuth, *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.

[111] A. Kolobov, "Planning with markov decision processes: An ai perspective," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–210, 2012.

[112] R. Kota, N. Gibbins, and N. R. Jennings, "Decentralized approaches for self-adaptation in agent organizations," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 7, no. 1, pp. 1–28, 2012.

[113] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 259–268.

[114] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, pp. 184–206, 2015.

[115] M. Kwiatkowska, "Quantitative verification: Models, techniques and tools," in *The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering: companion papers*. ACM, 2007, pp. 449–458.

[116] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 2.0: A tool for probabilistic model checking," in *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the*. IEEE, 2004, pp. 322–323.

[117] ——, "Quantitative analysis with the probabilistic model checker prism," *Electronic Notes in Theoretical Computer Science*, vol. 153, no. 2, pp. 5–31, 2006.

[118] ——, "Stochastic model checking," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer, 2007, pp. 220–270.

[119] ——, "Prism 4.0: Verification of probabilistic real-time systems," in *International Conference on Computer Aided Verification*. Springer, 2011, pp. 585–591.

[120] M. Kwiatkowska and D. Parker, "Automated verification and strategy synthesis for probabilistic systems," in *Automated Technology for Verification and Analysis*. Springer, 2013, pp. 5–22.

[121] P. Lalanda, J. A. McCann, and A. Diaconescu, *Autonomic computing*. Springer, 2013.

[122] J. Larsen and I. Pedersen, "Experiments with the auction algorithm for the shortest path problem," *Nord. J. Comput.*, vol. 6, no. 4, pp. 403–421, 1999.

[123] E. A. Lee, "Cyber physical systems: Design challenges," in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. IEEE, 2008, pp. 363–369.

[124] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: A cyber-physical systems approach*. Mit Press, 2016.

[125] H. W. Lenstra Jr, "Integer programming with a fixed number of variables," *Mathematics of operations research*, vol. 8, no. 4, pp. 538–548, 1983.

[126] P. R. Lewis, M. Platzner, B. Rinner, J. Tørresen, and X. Yao, "Self-aware computing systems," 2016.

[127] N. Li, C. Tsigkanos, Z. Jin, S. Dustdar, Z. Hu, and C. Ghezzi, "Poet: Privacy on the edge with bidirectional data transformations," in *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom*. IEEE, 2019, pp. 1–10.

[128] Z. Li, W. Li, and L. Jiang, "Task assignment problem of robots in a smart warehouse environment," *Management Studies*, vol. 4, no. 4, pp. 167–175, 2016.

[129] P. Maes, "Concepts and experiments in computational reflection," in *ACM Sigplan Notices*, vol. 22, no. 12. ACM, 1987, pp. 147–155.

[130] S. Malek, M. Mikic-Rakic, and N. Medvidovic, "A decentralized redeployment algorithm for improving the availability of distributed systems," in *International Working Conference on Component Deployment*. Springer, 2005, pp. 99–114.

[131] V. Matena, T. Bures, I. Gerostathopoulos, and P. Hnetynka, "Experimenting with adaptation in smart cyber-physical systems: A model problem and testbed," in *Engineering Adaptive Software Systems*. Springer, 2019, pp. 149–169.

[132] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. Cheng, "Composing adaptive software," *Computer*, vol. 37, no. 7, pp. 56–64, 2004.

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

[134] C. A. Micchelli, *Selected Papers Of Alan J Hoffman (With Commentary)*. World Scientific, 2003.

[135] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: a probabilistic model checking approach," in *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, 2015, pp. 1–12.

[136] ——, "Efficient decision-making under uncertainty for proactive self-adaptation," in *2016 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 2016, pp. 147–156.

[137] H. Muccini, R. Spalazzese, M. T. Moghaddam, and M. Sharaf, "Self-adaptive iot architectures: An emergency handling case study," in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, 2018, pp. 1–6.

[138] C. Müller-Schloer, H. Schmeck, and T. Ungerer, *Organic computing—A paradigm shift for complex systems*. Springer Science & Business Media, 2011.

[139] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[140] J. Nievergelt, "Exhaustive search, combinatorial optimization and enumeration: Exploring the potential of raw computing power," in *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer, 2000, pp. 18–35.

[141] J. Nievergelt, R. Gasser, F. Mäser, and C. Wirth, "All the needles in a haystack: Can exhaustive search overcome combinatorial chaos?" in *Computer Science Today*. Springer, 1995, pp. 254–274.

[142] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan *et al.*, "Ultra-large-scale systems: The software challenge of the future," DTIC Document, Tech. Rep., 2006.

[143] Parallel and M. U. Distributed Systems Laboratory. (2017) Yahoda - verification tools database. [Online]. Available: http://anna.fi.muni.cz/yahoda/

[144] M. Parashar and S. Hariri, "Autonomic computing: An overview," in *Unconventional Programming Paradigms*. Springer, 2005, pp. 257–269.

[145] D. Parker. (2017) List of verification tools database. [Online]. Available: http://www.prismmodelchecker.org/other-tools.php

[146] L. Perron and V. Furnon, "Or-tools," Google. [Online]. Available: https://developers.google.com/optimization/

[147] Y. Pochet and L. A. Wolsey, *Production planning by mixed integer programming*. Springer Science & Business Media, 2006.

[148] M. L. Powell and D. L. Presotto, "Publishing: A reliable broadcast communication mechanism," in *Proceedings of the Ninth ACM Symposium on Operating Systems Principles*, 1983, pp. 100–109.

[149] Q. Qiu, Q. Qu, and M. Pedram, "Stochastic modeling of a power-managed system-construction and optimization," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 20, no. 10, pp. 1200–1217, 2001.

[150] L. Ramshaw and R. E. Tarjan, "On minimum-cost assignments in unbalanced bipartite graphs," *HP Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2012-40R1*, 2012.

[151] P. Robertson and R. Laddaga, "Model based diagnosis and contexts in self adaptive software," in *Self-star Properties in Complex Information Systems*. Springer, 2005, pp. 112–127.

[152] M. Rubenstein, A. Cornejo, and R. Nagpal, "Programmable self-assembly in a thousand-robot swarm," *Science*, vol. 345, no. 6198, pp. 795–799, 2014.

[153] J. J. Rutten, *Mathematical techniques for analyzing concurrent and probabilistic systems*. American Mathematical Soc., 2004, no. 23.

[154] M. Salehie and L. Tahvildari, "Autonomic computing: emerging trends and open problems," in *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4. ACM, 2005, pp. 1–7.

[155] ——, "Self-adaptive software: Landscape and research challenges," *ACM transactions on autonomous and adaptive systems (TAAS)*, vol. 4, no. 2, p. 14, 2009.

[156] R. R. Schaller, "Moore's law: past, present and future," *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[157] E. Schmidt, "Technology is making marketing accountable," *Google Podium*, 2005.

[158] J. Schneider and S. Kirkpatrick, *Stochastic optimization*. Springer Science & Business Media, 2007.

[159] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang, "Cyber-physical systems: A new frontier," in *2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008)*. IEEE, 2008, pp. 1–9.

[160] M. Shaw, "Beyond objects: A software design paradigm based on process control," *ACM SIGSOFT Software Engineering Notes*, vol. 20, no. 1, pp. 27–38, 1995.

[161] P. Sinha and A. A. Zoltners, "The multiple-choice knapsack problem," *Operations Research*, vol. 27, no. 3, pp. 503–515, 1979.

[162] V. L. Smith-Daniels and D. E. Smith-Daniels, "A mixed integer programming model for lot sizing and sequencing packaging lines in the process industries," *IIE transactions*, vol. 18, no. 3, pp. 278–285, 1986.

[163] I. Sommerville, D. Cliff, R. Calinescu, J. Keen, T. Kelly, M. Kwiatkowska, J. Mcdermid, and R. Paige, "Large-scale complex it systems," *Communications of the ACM*, vol. 55, no. 7, pp. 71–77, 2012.

[164] G. Steel, "Formal analysis of pin block attacks," *Theoretical Computer Science*, vol. 367, no. 1-2, pp. 257–270, 2006.

[165] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland, "A concise introduction to autonomic computing," *Advanced Engineering Informatics*, vol. 19, no. 3, pp. 181–187, 2005.

[166] I. A. Şucan and L. E. Kavraki, "Accounting for uncertainty in simultaneous task and motion planning using task motion multigraphs," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 4822–4828.

[167] D. Sykes, J. Magee, and J. Kramer, "Flashmob: distributed adaptive self-assembly," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2011, pp. 100–109.

[168] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[169] G. Tel, *Introduction to distributed algorithms*. Cambridge university press, 2000.

[170] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White, "A multi-agent systems approach to autonomic computing," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 2004, pp. 464–471.

[171] O. Thapliyal and I. Hwang, "Path planning for a network of robots with distributed multi-objective linear programming," in *2021 American Control Conference (ACC)*. IEEE, 2021, pp. 4643–4648.

[172] G. Toffetti, A. Gambi, M. Pezzé, and C. Pautasso, "Engineering autonomic controllers for virtualized web applications," in *International Conference on Web Engineering*. Springer, 2010, pp. 66–80.

[173] R. J. Vanderbei *et al.*, *Linear programming*. Springer, 2015.

[174] P. Vromant, D. Weyns, S. Malek, and J. Andersson, "On interacting control loops in self-adaptive systems," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2011, pp. 202–207.

[175] S. Vukmirović, Z. Čapko, and A. Babić, "Model of using the exhaustive search algorithm in solving of traveling salesman problem (tsp) on the example of the transport network

optimization of primorje-gorski kotar county (pgc)," *JOURNAL OF CORPORATE GOVERNANCE, INSURANCE AND RISK MANAGEMENT*, 2019.

[176] M. Wallace, "Practical applications of constraint programming," *Constraints*, vol. 1, no. 1-2, pp. 139–168, 1996.

[177] D. Weyns, "Software engineering of self-adaptive systems," in *Handbook of Software Engineering*. Springer, 2019, pp. 399–443.

[178] D. Weyns and R. Calinescu, "Tele assistance: a self-adaptive service-based system exemplar," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 2015, pp. 88–92.

[179] D. Weyns, R. Haesevoets, and A. Helleboogh, "The macodo organization model for context-driven dynamic agent organizations," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 5, no. 4, pp. 1–29, 2010.

[180] D. Weyns, M. U. Iftikhar, D. G. De La Iglesia, and T. Ahmad, "A survey of formal methods in self-adaptive systems," in *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*. ACM, 2012, pp. 67–79.

[181] D. Weyns, M. U. Iftikhar, D. Hughes, and N. Matthys, "Applying architecture-based adaptation to automate the management of internet-of-things," in *European Conference on Software Architecture*. Springer, 2018, pp. 49–67.

[182] D. Weyns, M. U. Iftikhar, S. Malek, and J. Andersson, "Claims and supporting evidence for self-adaptive systems: A literature study," in *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 2012, pp. 89–98.

[183] D. Weyns, S. Malek, and J. Andersson, "On decentralized self-adaptation: lessons from the trenches and challenges for the future," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2010, pp. 84–93.

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

[185] W. Winston, "Introduction to mathematical programming: Applications and algorithms, duxbury,(2002)." 2002.

[186] J. S. Winter, "Emerging policy problems related to ubiquitous computing: Negotiating stakeholders' visions of the future," *Knowledge, Technology & Policy*, vol. 21, no. 4, pp. 191–203, 2008.

[187] L. A. Wolsey and G. L. Nemhauser, *Integer and combinatorial optimization*. John Wiley & Sons, 1999, vol. 55.

[188] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012.

[189] Y. Wu, "A mixed-integer programming model for global logistics transportation problems," *International Journal of Systems Science*, vol. 39, no. 3, pp. 217–228, 2008.

[190] L. Yang, J. Qi, and J. Han, "Path planning methods for mobile robots with linear programming," in *2012 Proceedings of International Conference on Modelling, Identification and Control*. IEEE, 2012, pp. 641–646.

[191] S. Yonbawi and R. Calinescu, "Towards self-adaptive systems with hierarchical decentralised control," in *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE, 2018, pp. 14–16.

[192] E. Yuan, N. Esfahani, and S. Malek, "A systematic survey of self-protecting software systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 8, no. 4, p. 17, 2014.

[193] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *ACM transactions on Software Engineering and Methodology (TOSEM)*, vol. 6, no. 1, pp. 1–30, 1997.