DECENTRALIZED SELF-ADAPTATION IN THE PRESENCE OF PARTIAL KNOWLEDGE

KISHAN KUMAR GANGULY
MSSE 0501

A Thesis
Submitted to the Master of Science in Software Engineering Program Office
of the Institute of Information Technology, University of Dhaka
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

Institute of Information Technology
University of Dhaka
DHAKA, BANGLADESH

DECENTRALIZED SELF-ADAPTATION IN THE PRESENCE OF PARTIAL KNOWLEDGE

KISHAN KUMAR GANGULY

Approved:

Signature *Date*

Student: Kishan Kumar Ganguly

Supervisor: Dr. Kazi Muheymin-Us-Sakib

To *Sankar Kumar Ganguly*, my father
whose endless inspiration has always kept me motivated

Abstract

Decentralized self-adaptive systems consist of multiple control loops that dynamically maintain the local goals of the locally managed systems and global goals of the whole system. As each locally managed system works together in a decentralized environment, all the control loops need to coordinate. That is, each control loop requires a global view of all the other ones. The seminal approaches in the literature are more concerned towards proposing generic decentralized self-adaptation frameworks. These assume a global view of the system and perform coordination in all the states to take adaptation decisions. However, Coordinating in such a way with global knowledge results in high coordination overhead. To reduce this overhead, some studies in the literature suggest that each control loop should coordinate with a limited set of control loops, which is termed as partial knowledge.

This thesis proposes a decentralized self-adaptation technique using reinforcement learning to incorporate partial knowledge and reduce coordination overhead. Optimal adaptation decisions are learned considering only the decision of single peer control loop. As global goals are the main source of coordination, to further reduce the overhead, coordination is done only in those states where global goals are violated. To support multi-objective self-adaptation, violated goals should be given more emphasis than the satisfied ones. So, a dynamic weight update scheme that provides more weights to the violated goals is also proposed.

The proposed approach was evaluated on a Tele Assistance System which is

a service-based exemplar and an adaptive robot navigation problem where four robots need to reach their goals avoiding collisions on the doorway. It was compared to three approaches that take decisions randomly, independently and jointly considering all the control loops. It was observed that, in all cases, it resulted in higher goal conformance and lower coordination overhead. The proposed approach was also compared with and without the weight update mechanism. The weight update technique was observed to provide higher goal conformance for all the goals where the other one led to goal violations. Overall, the proposed technique takes optimal adaptation decisions in the presence of partial knowledge with minimal coordination overhead.

Acknowledgments

I would like to thank and express gratitude to all the people who made this possible. Without the help and support from my supervisor, faculty and thesis committee members, classmates and my family, the thesis would not have been possible. I have learned a lot during this period of time from the people surrounding me.

I would like to express my gratitude to my supervisor Professor Dr. Kazi Muheymin-Us-Sakib for his supportive guidance, continuous inspiration and candid advice. His support and help have inspired me to relentlessly work for producing the best quality work I can. The regular feedback and constructive criticism helped a lot to view my work from different perspectives.

I would also like to appreciate the faculty and thesis committee members of Institute of Information Technology, University of Dhaka for their continuous feedback and helpful suggestions. Their valuable feedback, support and inspiration have greatly motivated me during my thesis.

I am grateful to my classmates for always supporting me. The time during the thesis was much pleasant due to their presence. I am greatly indebted to my parents who gave me hope to follow my passions up to this point.

I am also thankful to the Ministry of Posts, Telecommunications and Information Technology, Government of the Peoples Republic of Bangladesh for granting me ICT Fellowship No - 56.00.0000.028.33.079.17 (Part-1) -223 Date 20-06-2017.

List of Publications

1. "A Reusable Adaptation Component Design for Learning-Based Self-Adaptive Systems", in *Proceedings of the Twelfth International Conference on Software Engineering Advances (ICSEA)*, pp. 244-249, Athens, Greece, October 8 - 12, 2017.

2. "Decentralization of Control Loop for Self-Adaptive Software through Reinforcement Learning", in *Proceedings of the 1st International Workshop on Emerging Trends in Software Design and Architecture (WETSoDA) co-located with the 24th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 134-141, Nanjing, China, December 04, 2017.

3. "Decentralized Self-adaptation in the presence of Partial Knowledge with Reduced Coordination Overhead", in *Proceedings of The 10th International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE)*, Barcelona, Spain, 2018. (Accepted)

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the ever dynamic environment, software systems adjust itself continuously to respond to the context change. So, self-adaptation that enables to achieve this is a necessary requirement. Self-adaptive systems contain control loops which are the entities that manage self-adaptation. Although control loops can be both centralized and decentralized, with the growth of distributed systems, decentralized control loops are more preferable. The decentralized self-adaptive systems face more dynamism as multiple control loops need to coordinate continuously. That is, multiple control loops have to know the adaptation decisions of one another to work collaboratively. This hampers scalability, necessitating the need to work under partial knowledge, in other words, with a smaller set of known control loops. The coordination overhead also needs to be reduced for efficient adaptation. In this chapter, the motivation behind this work and challenges have been discussed. Issues in state of the art approaches are discussed next. This chapter also carries the research questions and contribution regarding the proposed decentralized self-adaptation mechanism. A section on how this thesis has been organized is mentioned at the end of this chapter.

1.1 Motivation

In this era of advanced computing, increased connectedness among systems has made decentralization an expected requirement. Currently, decentralization is also being considered in self-adaptive systems for better scalability and fault tolerance [1]. Fully decentralized self-adaptive systems are formed by adding a control loop to each component of the decentralized system which is termed as locally managed system in this thesis [2]. Each control loop manages some local goals related to the locally managed system. For example, maintaining a response time of maximum 2 seconds for the locally managed system is a local goal. The control loops altogether also manage some global goals which emerge from the local behaviors of the systems. For example, all the control loops may require maintaining a maximum system total response time of 4 seconds, which is a global goal. Satisfying these local and global goals continuously is the primary concern for a decentralized self-adaptive system [3].

As global behavior emerges from local behavior for decentralized self-adaptive systems, multiple control loops cannot decide independently of one another. In the worst case, all control loops may be interdependent. As a result, each needs to observe the decisions made by the other ones. This is not practical because providing each control loop with a global view harms scalability [3, 4]. So, each needs to decide in the presence of partial knowledge, which is challenging. It has also been seen that multiple dependent locally managed systems do not depend on one another in every state [5]. That is, some states require coordination and some states do not. The control loops can decide individually in these states where coordination is not needed, which reduces the coordination overhead. The challenge is to devise a self-adaptation technique that learns where coordination is required.

1.1.1 Motivational Example

The problems mentioned in the previous section are demonstrated here with two examples. The first one is a self-adaptive Tele Assistance System (TAS) proposed by Weyns et al. [4]. It is also a standard model problem from the Software Engineering for Self-Adaptive Systems exemplar repository [6]. The second one is an adaptive robot navigation problem used widely in the multiagent systems and decentralized self-adaptive system literature [5, 7, 8, 9].

Tele Assistance System (TAS)

Consider a Tele Assistance System (TAS) that provides medical service to people. Several vital parameters of the patients are measured. A Medical Analysis Service analyses these and invokes either an Alarm Service or Drug Service. Each of these services is supplied by individual service providers. Multiple variants of each of these services can be provided. Medical Analysis Service and Alarm Service have three variants each namely *MedicalAnalysisService1*, *MedicalAnalysisService2*, *MedicalAnalysisService3*, and *AlarmService1*, *AlarmService2*, *AlarmService3*. Drug Service has two variants which are *DrugService1* and *DrugService2*.

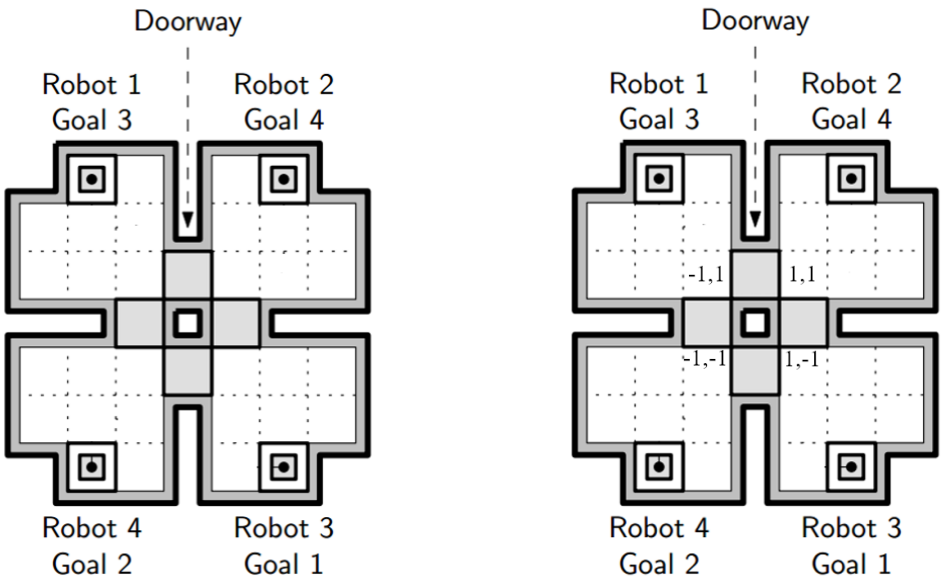
The local goals of TAS are listed below.

- Each service must have response time less than a local threshold
- Each service failure rate must be less than a specific local threshold

The following global goals are defined.

- At least one service must have failure rate less than a specific threshold
- The average cost per service must be minimized

These goals show that global goals emerge from local behaviors, that is, the satisfaction of local and global goals are related. For example, the failure rate local goal must be satisfied in such a way that the global one is also met. So, each



(a) Adaptive Robot Navigation inside a Grid

(b) Representation of the Grid in Cartesian Coordinate System

Figure 1.1: Example of Adaptive Robot Navigation inside a Grid

service control loop has to know the strategy of all the other service control loops. This is harmful to scalability because adding services in the system adds more coordination overhead. Another issue is that in states where the global failure rate goal is always met, coordination is unnecessary. It is rather needed in those states where global goals are violated.

Adaptive Robot Navigation

Figure 1.1 shows the adaptive robot navigation inside a grid. Four robots are placed in four corners of a grid. If the grid is represented in Cartesian coordinate like Figure 1.1(b), the initial positions of Robot 1, 2, 3 and 4 are (-2, 3), (2, 3), (2, -3) and (-2, -3) respectively. Each robot needs to reach the goal in the opposite corner of the grid. For example, Robot 1 has to reach Goal 1. The local goal is to reach the goal following the shortest path. The global goal is to avoid collisions in the doorway area (the gray area from the figure). Two robots cannot pass the doorway simultaneously.

In this case, the shortest path is the one with the minimum distance from the goal respecting the global goal (no collision). This requires four robots to know the route of one another. For example, consider the four robots arriving in the state similar to Figure 1.1(b). If Robot 1 wants to choose a path without collision, it needs to know the next step that the other three robots will take. Similar to TAS, adding more robots in the grid increases coordination overhead. The robots also need not coordinate in all the states. In the states near the doorway area, these must coordinate. In all the other states, coordination can be skipped as there is no chance for collisions on the doorway.

1.2 Issues in State-of-the-Art Approaches

State-of-the-art self-adaptation techniques are mostly for centralized control loops. RAINBOW framework proposed by Garlan et al. used probes to capture goal violations and propagated changes from an architectural model to the system through a Translation Infrastructure upon constraint violations [10]. However, the control loop in RAINBOW was not designed to support multiple coordinating control loops. The framework proposed by IBM had an autonomic manager (control loop) that provided adaptation capability to the managed system [11]. The autonomic manager itself contained sensors for being monitored and effector interfaces for changes to be applied. These could be utilized to add a meta-autonomic manager to it. Thus, multiple managers could hierarchically collaborate with one another. However, that did not allow for constructing a fully decentralized self-adaptive system with independent local control loops. Esfahani et al. proposed the FUSION approach that used learning to resolve uncertainty while using an optimization based solution to adaptation [12]. FUSION also did not explicitly assist decentralization because it did not consider coordination. Floch et al. proposed the MADAM framework where the system model was created and updated

at runtime to represent the most current representation [13]. Although it was not suited for decentralization, the concept of runtime update of model facilitated the quantitative verification techniques for decentralized self-adaptive systems.

The second research roadmap for self-adaptive system mentioned decentralization as one of the four major challenges where the other three were design space, generic process and practical run-time verification [14]. Weyns et al. discussed a generic model for designing decentralized self-adaptive system [4]. They mentioned four types of models namely system, concern, working and coordination models. System model reflected the core system where concern model contained its goals. Working model contained data structures and information needed to continue operation of MAPE (Monitor, Analyze, Plan and Execute) components [4, 11]. Coordination model was specific to decentralized control and contained coordination related information. However, this framework was generic and challenges mentioned previously were not addressed. Schmerl et al. proposed five patterns for decentralized self-adaptation which are coordinated control, information sharing, master-slave, regional planning and hierarchical control [1]. The first two of these correspond to fully decentralized control where only the information sharing pattern contains a shared information channel. The last three are partially decentralized with a centralized plan component and decentralized monitor, analyze and execute components of the MAPE model. Although they discussed the patterns with possible design characteristics, the aforementioned problems were not discussed.

A few rigorous approaches have been proposed for designing decentralized self-adaptive systems. Serugendo et al. discussed the state-of-the-art bio-inspired methods for decentralized self-organization [3]. They discussed several case studies such as managing computers networks with constraints, manufacturing control etc. to explain these methods [3]. These methods could not be generalized to self-adaptation and also did not incorporate partial knowledge. Nallur et al. pro-

posed a market-based decentralized self-adaptation technique for services in the cloud [15]. In their approach, buyer agents generated bids stating the Quality of Service (QoS) requirements of their required services while seller agents published asks mentioning the QoS capabilities of the concrete services. A market agent matched the bids with the asks using a multiple criteria decision making process. Although this approach is decentralized with multiple agents, the coordination of these agents, specially the concept of local and global goals was not presented. Sykes et al. proposed the gossip-based Flashmob approach to self-organization solving the problem of [16]. In this approach, each node transferred states to other nodes using the gossip protocol where a state is the global configuration that contained dependencies of all the components. To resolve these dependencies, each node chose the variant with the maximum local utility. However, the concepts of local and global goals were not present in Flashmob. Calinescu et al. proposed a method named DECIDE based on Markov chains and corresponding formal logic where the control loop required satisfying some local contribution-level agreement to reach global goal [17]. It achieved formal guarantees in conforming QoS level requirements. However, it is for mission critical systems specifically. It also does not consider how coordination between multiple MAPE loops can be designed.

Some Reinforcement learning-based approaches have also been proposed. Dowling et al. discussed a technique where the global optimization problem was broken down into similar multiple discrete optimization problems [18]. Then, each node either solved it by itself or delegated to its neighbour that can solve it more efficiently. Although minimization and maximization problems can be broken down in this way, threshold-based ones are much harder to break down. Wang et al. proposed the WSC-TMG Model based on joint Q-learning [19]. As coordination among all the control loops is required, it suffers from the problems mentioned previously. Caporuscio et al. proposed an approach in [20] for self-adaptive service composition where gossip protocol was used to transmit service configurations. A

primary list of dependency resolving services was selected with maximum utility similar to [21]. A second-level scanning was performed using learned reward values from Q-learning, which provided the final dependency resolving services. However, it was not explicitly mentioned whether joint Q-learners or individual Q-learners were used. Besides, The technique to calculate global reward functions required dissemination of each local reward function values to each of the agents.

It is evident from the discussion that the recent endeavors in building decentralized self-adaptive systems are mostly either generic or very specific to certain types of systems. Although some approaches have the potential to develop decentralized self-adaptive systems, most of these do not consider coordination overhead and partial knowledge incorporation. So, the literature indicates that decentralization of control loop for self-adaptive systems has much scope for improvement.

1.3 Research Questions

Partial knowledge management and reduction of overhead is a crucial requirement for multiple coordinating control loops in decentralized environment. Current methods in the literature mostly focus on establishing a generic framework rather than these. Some techniques focus on efficiency and scalability, but these issues are ignored. Decentralized self-adaptation using partial knowledge is challenging because, in some states, all the control loops are truly dependent on one another. Coordination overhead reduction is also difficult because coordination is the core element of decentralized self-adaptive systems. All these lead to the following research question.

1. How to incorporate partial knowledge and reduce coordination overhead in decentralized self-adaptive systems?

   More specifically, this research question can be answered by the following sub-questions.

(a) How to achieve optimal solution (satisfaction of local and global goals) with partial knowledge?

To answer this sub-question, following steps may be adopted:

i. The coordination mechanism can be designed with reinforcement learning algorithms.

ii. During coordination, only one agent can be used as a peer. A learning algorithm that helps to achieve this can be designed.

(b) How can coordination overhead be minimized?

Literature shows that coordination is not required in all the states [7]. These states can be learned from the rewards produced by the environment. So, the coordination overhead will be reduced.

1.4 Contribution and Achievement

This work makes the following contributions.

1. A reinforcement learning-based technique for decentralized self-adaptation is proposed that incorporates partial knowledge assumption. An Interaction Driven Markov Game (IDMG) [5] based technique is proposed that enables using a single peer to take adaptation decisions.

2. A technique to learn states where coordination is beneficial is introduced.

3. An approach to support multi-objective self-adaptation is presented.

Q-learning under the Interaction-Driven Markov Game (IDMG) model is utilized rather than using Q-learning based on the traditional Markov game model [5]. Local and global reward functions are defined which calculate local and global goal conformance respectively. The states where global goals are always satisfied but local goals may be violated are regarded as non-coordinating states. This is

because global goals are directly related to coordination and so, their continuous satisfaction indicates that coordination may not be required. Here, individual Q-learners are used for self-adaptation [22]. In states where global goals may not be satisfied, coordination is considered under IDMG-Based Q-learning [7]. In this case, each local control loop coordinates with a single peer rather than multiple peers. Melo et al. showed that considering only a single peer while performing such type of learning does not significantly reduce goal conformance [7]. Thus, the proposed approach addresses both partial knowledge and coordination overhead. Multi-objective self-adaptation is supported by using a weighted sum of all the reward functions. The weights of the violated goals should be more than those of the satisfied goals. So, these weights are dynamically updated as goals are violated.

The approach was validated on the aforementioned two systems - adaptive robot navigation and TAS. Average rewards over multiple runs were compared among the proposed approach, individual Q-learning and joint Q-learning (Q-learning requiring all the control loops to coordinate). It was observed that the proposed technique resulted in much higher reward which indicates higher effectiveness. The learning time steps are were also compared which showed that the proposed technique converges much faster. The weight update technique was validated by executing the proposed approach once with the dynamic weight update and again without any weight update, and then comparing the rewards of all the runs. With weigh update, all the goals were continuously satisfied. Without weight update, some goals were violated during runs. The results overall show that the proposed approach can effectively learn optimal adaptation decisions considering a single peer. The technique converges much faster even after coordinating only in specific states to reduce coordination overhead. The approach also successfully performs multi-objective self-adaptation in the decentralized environment.

1.5 Organization of the Report

In this section, the organization of the report has been shown to provide a roadmap to this document. The organization of the chapters in this report is as follows.

Chapter 2: The definitions and background information of decentralized self-adaptive systems have been discussed in this chapter. The definition and general frameworks of self-adaptive systems have been described. Decentralized self-adaptation has been defined with examples. The challenges in this domain along with the solution concepts are also discussed in this chapter.

Chapter 3: In this chapter, the existing works for self-adaptive system design have been presented is a structural way. The literature is classified into four parts namely general frameworks and design, rule-based, biologically inspired and learning-based. Seminal works from each of these are discussed.

Chapter 4: This chapter contains the proposed methodology for the decentralized self-adaptation. The problem is formulated and the definitions of state, actions and reward functions are provided in the context of the proposed solution. The learning technique is also provided along with the dynamic weight update mechanism. This chapter uses the aforementioned examples of adaptive robot navigation and TAS to explain the methodology.

Chapter 5: The experimental setup, implementation and result analysis based on the two examples are presented in this chapter.

Chapter 6: This is the chapter that summarizes the whole thesis and highlights future work.

Chapter 2

Background Study

In this era of advanced technology, smart cities, Internet of Things, unmanned vehicles etc. are moving towards more connectedness. Due to this, the operating environment of a software has become overwhelmingly complex [23]. So, environment changes are recurrent. Software must change its behavior with the changing environment, otherwise, the altered context will lead to requirement violation. Hence, this self-adaptive behavior or self-adaptation is a desirable property in the ever dynamic environment. With the growth of decentralization, self-adaptive systems should be decentralized as well. This is because decentralization offers scalability, fault-tolerance etc. which enable large-scale application of self-adaptation [4]. Thus, A fully decentralized self-adaptation can substantially extend its application domain. Moreover, it can also improve the scalability and efficiency in the current application areas of self-adaptive systems such as service-based systems, traffic control etc. Overall, decentralized self-adaptation is a concept that must be exercised to develop highly dynamic software with connectedness. This chapter contains the Background information to understand decentralized self-adaptive systems. The definition and properties of self-adaptive systems are discussed first. Some general frameworks of self-adaptive systems are given next, followed by the definition and examples of decentralized self-adaptive systems. Finally, solution

concepts in this domain such as rule-based, learning-based etc. are described in the context of the literature.

2.1 Self-Adaptive System

Self-adaptive systems are those that continuously change their behavior at runtime in response to environment changes. The key aspect of self-adaptation is variation which is also called modelling dimension in the literature [24, 25]. Cheng et al. mentioned four modelling dimensions which are goal, change, mechanisms and effect [24]. These are briefly described below.

- Goal: A goal is a measurable objective that needs to be reached. For example, consider a software with a goal of maintaining a maximum 5ms response time. If this limit crosses at any point, a goal violation is said to have occurred.

- Change: This is the reason behind adaptation. Context change triggers the adaptation process. Context change can be caused by the actors (who interact with the system), the environment (the external world with which the system interacts with) and the system itself. For example, a large number of concurrent users may lead to the violation of the aforementioned 5ms response time goal. Here, context change is triggered by actors.

- Mechanism: This is the response of the self-adaptive system towards context change and goal violation. A few aspects of this need to be discussed which are type, organization, scope, and duration. Type of adaptation can be parametric or structural. Parametric adaptation happens when the system responds by changing some of its parameters (e.g., maximum connection limit, buffer size etc.). Structural adaptation happens when system components are reorganized for self-adaptation. In the previous example, a faster

version of some modules with minimal features can be used when performance goal violation occurs. Organization of adaptation can be centralized or decentralized. A single component performs adaptation for the centralized one and multiple coordinating components perform adaptation for the decentralized one. The scope of adaptation covers whether adaptation is localized or globalized. Local adaptation deals with particular components only where global adaptation deals with the full system. Duration describes the time length for adaptation. In most cases, adaptation duration should be short.

- Effect: It describes the impact of adaptation on the system. It has several sub-dimensions which are criticality, predictability and overhead. Criticality of adaptation means the impact of a failed adaptation. A failed adaptation may be tolerated in some systems where it may not be accepted in safety-critical systems. Predictability denotes whether impact can be predicted beforehand. Overhead shows the performance degradation of adaptation. Overhead can range from minimal to thrashing where the system performs more work for adaptation than its usual functions.

Self-adaptation also has some properties, know as Self-* properties that depict how self-adaptation may look like. IBM described four properties which are self-configuring, self-healing, self-optimizing and self-protecting [26]. Self-configuring refers to reconfiguring components automatically in response to changes. Self-healing entails finding and diagnosing issues and resolving these. Self-optimizing systems manage and allocate resources in such a way that performance is optimal according to the requirements. Self-protecting means the automated detection of security vulnerabilities and attacks and recovering from these.

Figure 2.1 shows an example of a self-adaptive system. The adaptation component is the external self-adaptation manager and the business logic components

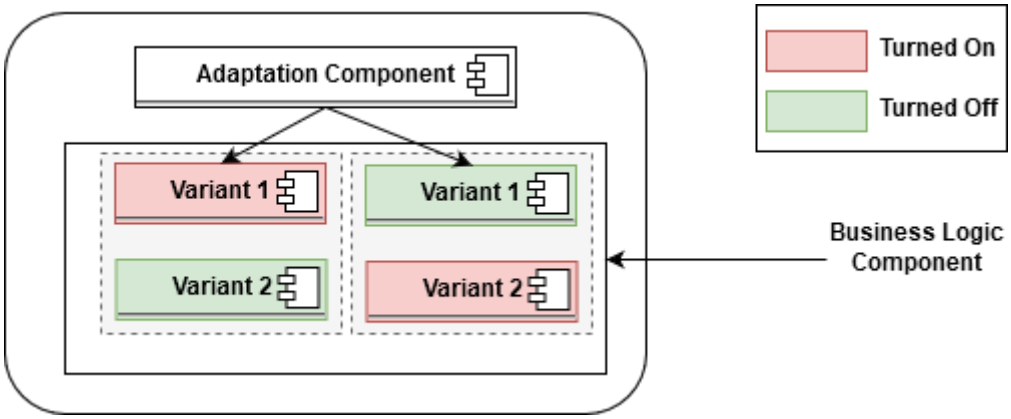


Figure 2.1: An Example of a Self-Adaptive System

are of the managed system. The managed system contains variants (similar functionality components with varying configurations) of all the components. So, this is an example of structural adaptation. The adaptation component monitors the system by observing goal values. Goal violation triggers adaptation mechanism. It plans for adaptation by analyzing the violated goal values and executes the plan by reconfiguring the components (turning on or off variants).

From the definitions and the examples provided above, self-adaptive systems work in a continuous loop of four steps which are monitor, analyze, plan and execute [11]. Firstly, *monitor* helps to observe itself by extracting goal values at runtime. Secondly, *analyze* scans monitored values for goal violations. Thirdly, *plan* finds the satisfactory configuration. Finally, *execute* applies the calculated configuration to the system. This constitutes a feedback or control loop which IBM states as the MAPE-K architecture [11].

2.2 MAPE-K Architecture

The *MAPE-K* architecture was proposed as a blueprint for an *autonomic manager* [11, 27]. According to IBM, an autonomic manager is a component that automates a management function and makes this function externally visible using an interface [11]. The architecture of this autonomic manager has been shown

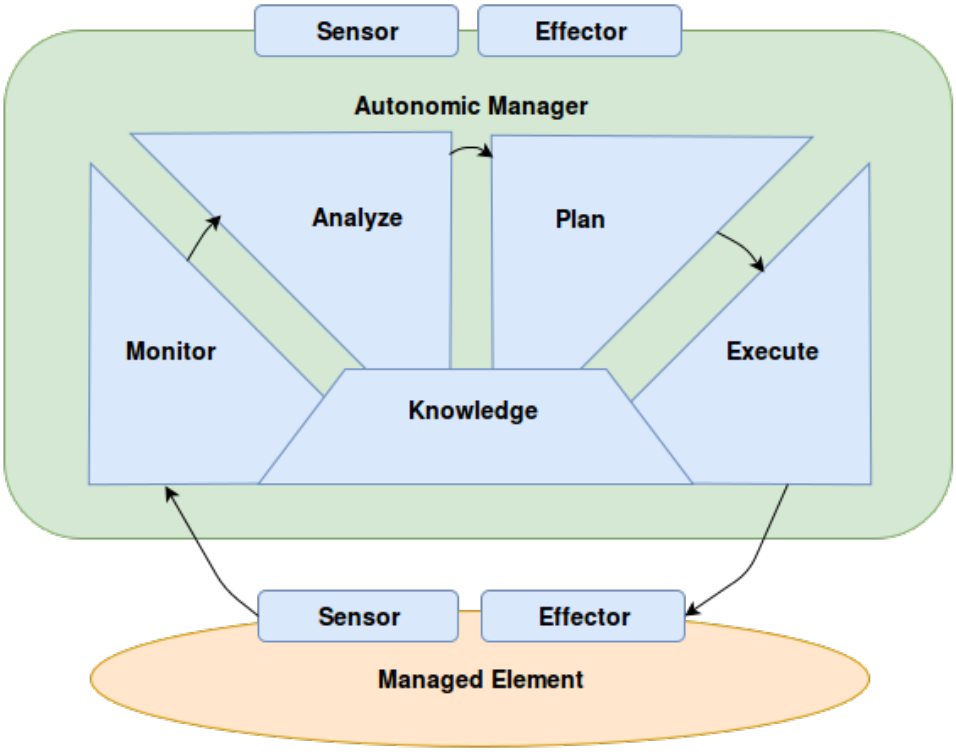


Figure 2.2: MAPE-K Feedback Loop by IBM

in Figure 2.2. The figure depicts that information captured by sensors are passed to *monitor* function and then it is inspected by *analyze* function. The analysis triggers the *plan* function and finally plans are executed through effectors from the *execute* function. Each part is described briefly below.

1. Monitor: This function receives information from the sensor interface. The information is aggregated or organized in such a way that these correspond to analyzable symptoms. For example, collected information can be mapped to a metric (for example, performance) and passed to the analyze function.

2. Analyze: This determines if symptoms passed from the monitor indicate a goal violation. In this case, a logical change request for that particular symptom is passed to the plan function. For example, if performance degradation is detected, a change request indicating performance goal violation is issued.

3. Plan: The activities to return the system close to its goal are developed by

the plan function. These activities can be as simple as a single command or complex such as a workflow consisting of multiple activities. In the example of performance degradation, a plan can be constructed to disable some memory consuming components that have been unused for a particular time period.

4. Execute: This function schedules and performs changes to the system. These changes are done through the effector interface [11].

5. Knowledge: Knowledge is the data used and shared between the four functions mentioned above. It includes policies, logs, historical information etc. A knowledge base can be pre-supplied or constructed by the autonomic manager. The historical information in the knowledge base needs to be updated regularly as it holds the patterns useful for predicting resources or symptoms.

The knowledge can also be divided into three types which are *solution topology knowledge*, *policy knowledge* and *problem determination knowledge* [11]. Solution topology knowledge consists of system configuration and component information which the plan function may use for choosing valid activities. Policy knowledge is used for deciding if changes can be deployed into the system. Problem determination knowledge consists of monitored and symptoms data that the loop may use to learn behaviors of adaptation.

2.3 Three Layer Conceptual Model

Apart from MAPE, another famous architectural model for self-adaptation is the three layer conceptual model by Kramer et al. [28]. The development of this model was inspired by models in autonomic systems, specially, robotics [29, 30]. The three layers of this models are component control, change management and goal management layers. Each of these layers has specific responsibilities and asks

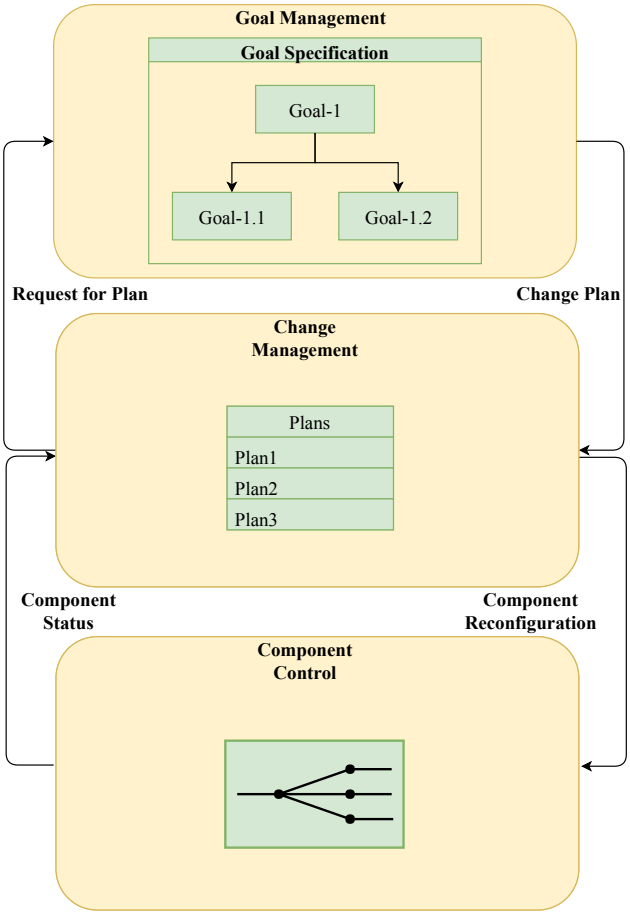


Figure 2.3: Three Layer Conceptual Model

for service from its immediate upper-level layer. Figure 2.3 shows this model. The three layers are described below.

- Component Control: The component control layer contains specific component configurations and interfaces for adding, removing and modifying components. This layer is capable of tuning the component parameters for adaptation. However, it cannot reorganize components. When a goal cannot be met by only parameter optimization, it sends the component status to the change management layer.

- Change Management: This layer contains some preconfigured plans. When it receives the component status from the component control layer, it analyzes the goal values and current component configuration. It then adds,

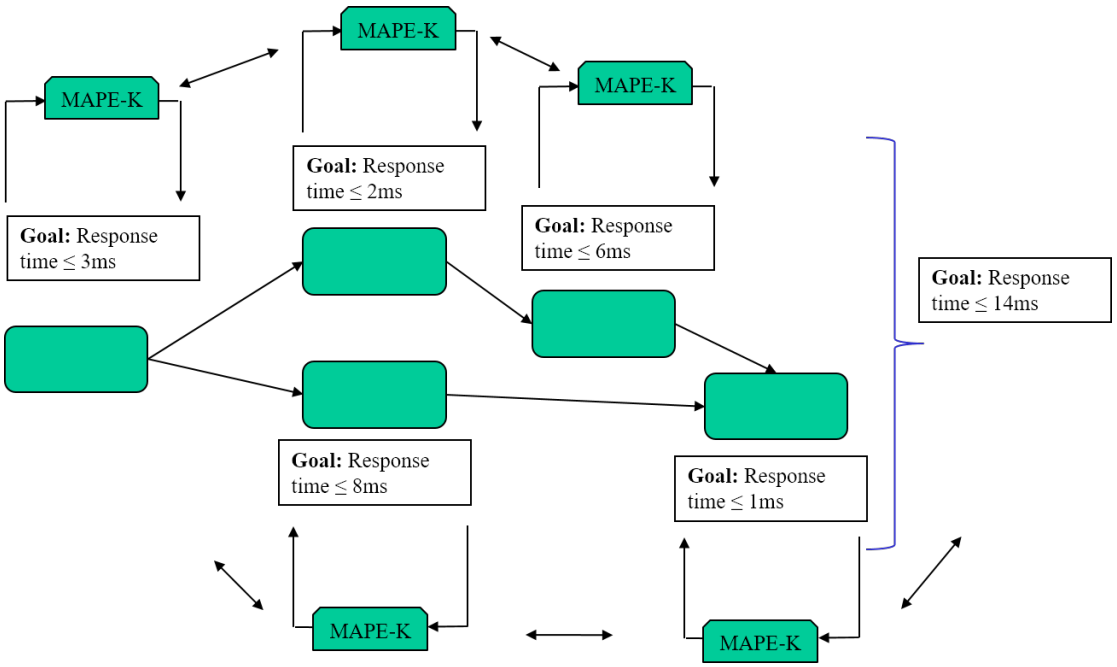


Figure 2.4: An Example of Fully Decentralized Self-Adaptive System

removes or modifies components (i.e., reconfiguration) through the component control layer interface. If a goal cannot be satisfied using the existing plans in this layer, it requests the goal management layer for generating plans.

- Goal Management: The main responsibility of the goal management layer is to generate plans due to change management layer request or new goal introduction. For example, consider that the change management layer fails for the goal of maximum service response time 5ms when the number of concurrent users exceeds 2000. Goal management layer analyzes this issue and finds a plan for this specific condition. This plan is sent to the change management layer.

2.4 Decentralization of Control Loop

The decentralization of control loop is necessary where multiple control loops need to coordinate for adaptation. The control can be hierarchically connected so that

one controller adapts the behavior of another one [1, 4]. The example of this type of control scheme is IBM's MAPE-K blueprint [11]. An autonomic manager provides sensors and effectors itself to be managed by another orchestrating autonomic manager. Generally, the autonomic manager at the lower hierarchy level has the responsibility to adapt the managed resources where a higher level autonomic manager controls system-wide adaptation [4]. However, the hierarchical control scheme has two drawbacks. Firstly, the failure of a particular autonomic manager in the hierarchy means the loss of a certain level of behavior. Secondly, scalability is still compromised as lower level autonomic managers manage system-level resources in a centralized way. These two issues are solved by fully decentralized autonomic managers or control loops [4]. In this case, control loops act as peers. Each of these provides autonomic capabilities to a subsystem with some *local goals*. All the control loops also need to satisfy some *global goals* which are the system-wide behaviors. However, the satisfaction of global goals emerges from the satisfaction of local goals through interactions [14, 31].

Figure 2.4 shows an example of local and global goals in fully decentralized control loops. The subsystems are managed by control loops which have different response time goals expressed in thresholds. Each control loop needs to maintain the response time dynamically. However, the overall system response time also needs to be under 14ms which is the global goal (Figure 2.4). These control loops must choose their configuration in such a way that both the local and the global goals are satisfied. This demands coordination where each local control loop may choose a less optimal solution for the satisfaction of other control loop local goals and the global goals.

The IBM's MAPE-K blueprint does not directly support autonomic managers to be acting as peers. Centralized solutions that are present in the literature also cannot be trivially extended to support such capability [4]. Some centralized self-adaptation techniques are described here. RAINBOW framework proposed by

Garlan et al. used probes to capture goal violations, propagated changes from an architectural model to the system through a Translation Infrastructure upon constraint violations [10]. However, the control loop in RAINBOW did not support multiple coordinating control loops. The framework proposed by IBM had an autonomic manager that provided adaptation capability to the managed system [11]. The autonomic manager itself contained sensor and effector interfaces that could be utilized to add a meta-autonomic manager to it. Thus, multiple managers could hierarchically collaborate with one another. However, a fully decentralized self-adaptive system with independent local control loops was not supported. Esfahani et al. proposed the FUSION approach that used learning to resolve uncertainty while using an optimization based solution to adaptation [12]. FUSION also did not explicitly assist decentralization because it did not consider coordination and partial knowledge. Floch et al. proposed the MADAM framework where the system model was created and updated at runtime to represent the most current representation. Nevertheless, this also did not provide decentralized capabilities [13].

Designing self-adaptive systems with decentralized control loops is highly challenging. These challenges occur from mainly two facts in these types of systems. Firstly, the local control loop does not have a complete view of all the local control loops. If the local control loops could track the goal violations and decision of other local control loops, it could have derived an optimization problem to optimize itself considering other control loop local goals and global goals [32]. However, this is not possible due to the partial knowledge of the local control loop that enables itself to observe only a limited set (generally neighbours) of all the control loops. Secondly, the coordination overhead should be minimal. It is notable that incorporating partial knowledge already reduces the coordination overhead to some extent. So, consideration of partial knowledge complements reducing coordination overhead.

2.5 Application Domain of Decentralized Self-Adaptive Systems

Although any distributed system can be augmented with decentralized control loops, a several application areas are the most important ones such as traffic control, robotics, service-based systems, adaptive routing, distributed unmanned underwater vehicles, Internet of Things (IoT) etc. These are described as follows.

- Traffic Control: A popular use of decentralized self-adaptive systems is traffic light control. Multiple cameras monitor a specific area for traffic jam. When multiple cameras detect traffic jams in nearby areas, these collaborate to appropriately turn on and off traffic lights. Weyns et al., Dusparic et al. and Wedde et al. discussed such techniques in [4] [33] and [34] respectively.

- Robotics: Another area where decentralized self-adaptive systems are used is robotics. For example, it is widely used for robot navigation where multiple robots need to reach their goals avoiding any physical conflict [5, 7, 35]. It is also used in the problem of carrying objects from one place to another through robotic arms. Here, multiple robotic components may be reconfigured to dynamically support some nonfunctional properties. Examples of this include self-assembly of robotic arm components by Sykes et al. and self-adaptive robots by Yu et al. [36, 37].

- Service-Based Systems: Decentralized self-adaptive systems are common in the field of service assembly. In this case, multiple services need to be dynamically composed for performing a specific task. The most optimal components are required for the satisfaction of the local and the global goals. The dynamic service composition approach by Wang et al., SAFDIS approach by Gauvrit et al. are some of the examples of such systems [19, 38].

- Adaptive Routing: This type of routing exploits both local knowledge in a

router and knowledge received from the peer to make routing decisions. The routing decision may require selecting the shortest routing path, the highest performing routing path etc. These requirements lead to solving local and global optimization problems by the routers. Dowling et al. discussed such an approach named adhoc routing [18]. Some other examples of this include adaptive MANET routing by Dowling et al. and self-adaptive multi-path routing in overlay networks by Leibnitz et al. [39, 40].

- Distributed Unmanned Underwater Vehicles: Unmanned underwater vehicles are used for surveillance in the ocean, mine detection, rescue operation etc. [17]. Distributed unmanned underwater vehicles are used where large-scale surveillance is required. Multiple vehicles have different local goals and common global goals. For example, in the DECIDE approach for decentralized self-adaptation by Calinescu et al., they define a local goal stating the accuracy probability of the sensor must be above a specific local threshold [17]. In this approach, they also specify global goals, for example, at least two vehicles must have turned on sensors at the same time [17]. Decentralized self-adaptation is necessary to satisfy these goals at runtime.

- IoT: With the growth of IoT, self-adaptiveness is also being introduced into this domain. Vermesan et al., in their research roadmap, mentioned that introducing self-* properties is one of the most important research challenges in IoT, specially during 2015-2020 [41]. Nascimento et al. discussed the smart city as a realization of decentralized self-adaptive system concepts [42]. Similar to traffic control, sensors are present in smart cities that collect various types of data such as traffic information, water and energy consumption etc. [42]. These data are used to optimally distribute resources and perform actions such as traffic light control, energy production control etc. In an ACM SIGSOFT webinar, Danny Weyns also mentioned the im-

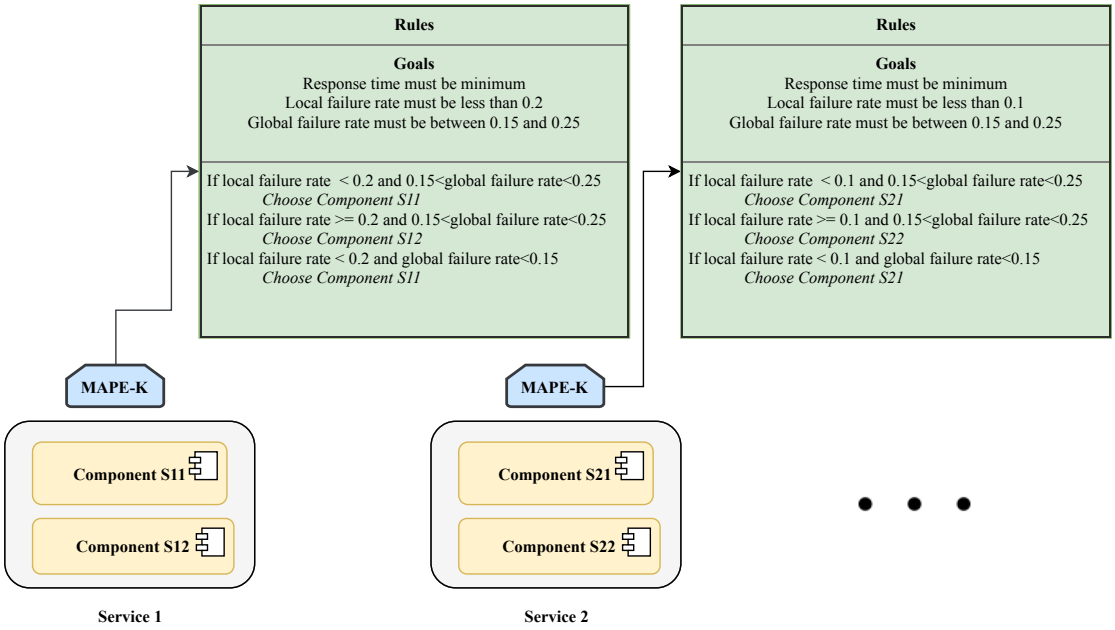


Figure 2.5: An Example of Rule-Based Decentralized Self-Adaptive System

portance of introducing decentralized self-* properties in building IoT-based smart cyber-physical systems [43].

2.6 Solution Concepts in Decentralized Control for Self-Adaptive Systems

In the literature, solution to decentralized control has been approached from multiple viewpoints such as static rule-based [44], learning-based [18, 19, 33], biologically inspired [45] etc. In this section, these approaches are discussed. The mechanism behind rule-based techniques is described first with an example. Some definitions and properties of biologically inspired approaches are explained. Learning-based approaches are discussed in details. Moreover, the popular gossip approach for information dissemination in decentralized self-adaptive systems is described.

2.6.1 Rule-Based

In rule-based approaches, condition-action rules are used to satisfy local and global goals. For example, consider the decentralized self-adaptive service assembly problem in Figure 2.5. Each service chooses the component that satisfies some local and global goals prior to service composition. The service component selection rules are specified according to the goals. For example, the first rule states that if failure rate local goal and global goal both are satisfied, the highest performing but less reliable Component S11 is chosen. However, if local failure rate goal violation occurs, it is prioritized over performance goal satisfaction and the less efficient but more reliable Component S12 is selected. Thus, in every local MAPE-K loop, the plans are hardwired.

The obvious benefit of this approach is its simplicity. However, there are several drawbacks. Firstly, the number of rules depends on goals, system states, environment predictability etc. For a large number of goals, devising large number of rules manually is difficult. Besides, the state space of the system may also be large and the environment is often highly dynamic. These make the rule-based approach less appealing in real-life decentralized self-adaptive system development. Secondly, if the dynamics of the environment change, these rules become invalid. In this case, new rules need to be derived which is a time-consuming process. For these reasons, rule-based solutions are not suitable for large-scale systems. If the system has a few goals, the environment is stable and the state space is small, the rule-based technique is effective.

2.6.2 Biologically Inspired

These are approaches that are based on principles of natural life. Biologically inspired approaches are most appropriate for self-organization. This is a process where system components organize optimally by themselves [31]. A biologically

inspired decentralized self-adaptive approach should support the following principles [31].

- Noise: This is some form of disturbance that is added to the technique for reaching the global optimum. If noise is not added, the technique may get stuck into a local optimum. An example of noise is the mutation phase in the genetic algorithm [46].

- Emergence: This is the most important aspect of these types of solutions. Emergence indicates that global goal satisfaction appears as a result of local goal satisfaction. In the biological world, an example of emergence is swarming of bees, birds etc. [47]. Each entity follows the simple local rule of following the direction of the neighbour while avoiding conflict. This simple rule, when followed by thousands of entities, create complex group shape and behavior. A similar situation is seen in ant colonies [48]. In case of decentralized self-adaptation, from Figure 2.5, the satisfaction of local failure rate goal must lead to global failure rate goal satisfaction.

- Diffusion: Diffusion is a way to spread information fast [31]. Each piece of information sent to an entity is forwarded to another one and this process is continued until the whole group has received this information. Gossip is a diffusion-based approach that works in this way (discussed in Section 2.6.4).

- Stigmergy: This is another method for communication. In this case, rather than directly communicating the information, it is left in the environment for other entities to discover.

- Evolution: To get better result as time progresses, evolution should occur. Evolution chooses the highest performing variants and removes the lowest performing ones over time. Coevolutionary algorithms are used to build multiagent systems exploiting this concept [49, 50, 51]. As coordination is

supported in this technique, it can also be considered to build decentralized self-adaptive systems.

2.6.3 Learning-Based

Although centralized self-adaptive systems have been approached using both supervised and reinforcement learning, decentralized ones mostly use reinforcement learning. A well-known branch of reinforcement learning, called multiagent Q-learning is widely used in the literature. Multiagent Q-learning is closely related to game theory, more specifically Markov game. These concepts are presented in the next subsections.

Game Theory

Game theory models strategic interaction between a set of players. Each player receives a reward or payoff when they participate in the game due to their interactions. A game can be represented in a game matrix. Table 2.1 shows a game matrix.

Table 2.1: Game Matrix for A General Sum Game

		Player Y	
		A	B
Player X	A	$(5, 5)$	$(0, 10)$
	B	$(10, 0)$	**(1,1)**

Here, for example, cell (A, B), the first reward 0 corresponds to the reward received by player X and the second reward 10 corresponds to the reward received by player Y. Here, the bold rewards is the *Nash equilibrium* for this game [52]. Nash equilibrium corresponds to the strategy where no player can improve his payoff by unilaterally deviating from this strategy [53]. For example, consider Player Y chooses A, the best response by Player Y is certainly B. Again consider Player X chooses B, in this case, Player Y will choose B also. In the opposite

scenario, Player X will choose B whether player Y chooses A or B. That is, B is the best strategy a player can follow without knowing the other player's option. In case of a decentralized self-adaptive system, if control loops are considered as players, the target is to reach the Nash equilibrium. Solution strategies such as multiagent Q-learning algorithms can guarantee to converge to a Nash equilibrium [22, 54].

Multiagent Q-learning

Q-learning problems considering multiple agents (locally managed systems) can be described as a Markov game [53]. A Markov game is defined as follows.

Definition 2.1. A Markov game is defined by a tuple $(n, S, A_{1...n}, R_{1...n}, T)$ where,

- n is the number of agents or locally managed systems that cooperate.
- S is a finite set of states.
- A_i refers to the action available to the i^{th} agent.
- $R_i : S \times A_1 \times A_2 \times ... \times A_i \times ... \times A_n \to \mathbb{R}$ refers to the reward function of the i^{th} agent when transitioning from one state to another applying a specific joint action.
- $T : S \times A_1 \times A_2 \times ... \times A_i \times ... \times A_n \to P(S)$ is the transition function when transitioning from a specific state applying a specific joint action. Here, $P(S)$ is a function that assigns probability to the states.

In a Markov game, each agent follows a playing strategy. A strategy is an action selection rule in a specific state. When a state transition occurs after an action, each agent receives a reward according to its reward function. The goal is to find the strategy that maximizes the rewards of the agents. Q-learning helps to maintain and continuously update a long-term reward value for each state and

Algorithm 2.1 Algorithm for Multiagent Q-learning

1: $Q_i(s,a) \leftarrow 0, \forall s,a,i$
2: $decay \leftarrow d_f$
3: **while** $TerminationCondition \neq true$ **do**
4: **for** $i = 1$ to n **do**
5: $a_i \leftarrow selectAction()$
6: $a^{-i} \leftarrow receiveOtherAgentActions()$
7: $a \leftarrow a_i \cup a^{-i}$
8: *observe transitioned state* s' *and reward* r_i
9: **for** $i = 1$ to n **do**
10: $Q_i(s,a) \leftarrow Q_i(s,a) + \alpha \times [r_i(s,a) + \gamma \max_{a_i} Q_i(s',a_i) - Q_i(s,a)]$
11: $\alpha \leftarrow \alpha - decay \times \alpha$
12: **end for**
13: **end for**
14: **end while**

joint action which is called the Q-value. Each agent maintains its own Q-values for different states and joint actions. During action selection in a specific state, it chooses the action that maximizes the Q-value.

Algorithm 2.1 shows the Q-learning technique [19, 55]. After action selection and state transition, each agent observes its reward (line 8). Each of the Q-values for the agent is updated with its immediate reward $r_i(s,a)$ and Q-value of the best action a_i that the agent can select in the next state (line 10). However, as Q-values are maintained for joint actions only, maximum attainable state Q-value by an agent depends on the strategies of the other agents. So, an estimation of other agent strategies is needed to calculate the next state maximum Q-value. This is achieved using Equation 2.1.

$$Q_i(s,a_i) = \sum_{a^{-i} \in A_c^{-i}} P(s,a^{-i}) \times Q_i(s,a_i,a^{-i}) \tag{2.1}$$

$$P(s,a^{-i}) = \frac{F(s,a^{-i})}{\sum_{a^{-i'} \in A_c^{-i}} F(s,a^{-i'})} \tag{2.2}$$

Equation 2.1 sums the Q-values considering all possible actions of the other agents ($a^{-i} \in A_c^{-i}$) weighted by the action probabilities. The $Q_i(s,a_i)$ denotes the

expected Q-value of an individual agent. The action probability is calculated using Equation 2.2. $P(s, a^{-i})$ is called the opponent model which is maintained by each agent for each possible other agent actions a^{-i} [53]. Thus, Q-value calculation of an agent incorporates dependency on other agents' action selections. This technique is also known as fictitious play and is a popular research area in game theory [56, 57].

The Q-value update equation from Algorithm 2.1 introduces two parameters which are α and γ. These are the learning rate and the discount factor respectively. Learning rate controls how much the learned value weights against the current value. In the beginning, learning rate should be higher because the Q-value table is empty. The α should be decayed by a prespecified decay factor in every iteration to continuously reduce the amount of learning. This helps an experienced agent to believe in the already known Q-value rather than the recently acquired knowledge. It also ensures that the algorithm converges (i.e., Q-values do not change for further iterations). γ balances the weight between the immediate reward and the future reward. It means that γ controls how far to look into the future when calculating reward. Both α and γ are set to values between 0 and 1 [58].

When an agent chooses an action a_i in a state s, it calculates $\max_{a_i} Q_i(s, a_i)$ using Equation 2.1 and 2.2 for this purpose. However, along with this exploitative strategy, exploration (i.e., choosing actions randomly) also needs to be performed occasionally. This is because occasional exploration helps to search for actions with better reward than the current one. Out of many exploration strategies, a popular one is the $\epsilon - greedy$ strategy because it is simple but effective [22]. According to this exploration technique, an ϵ value is prespecified where $0 \leq \epsilon \leq 1$. During action selection, a value p between 0 and 1 is randomly selected. If $p < \epsilon$, the agent chooses a random action without considering the learned Q-values. Otherwise, it selects the action obtained from the aforementioned exploitative strategy (i.e., using the Q-values). It means that random action is chosen with probability ϵ

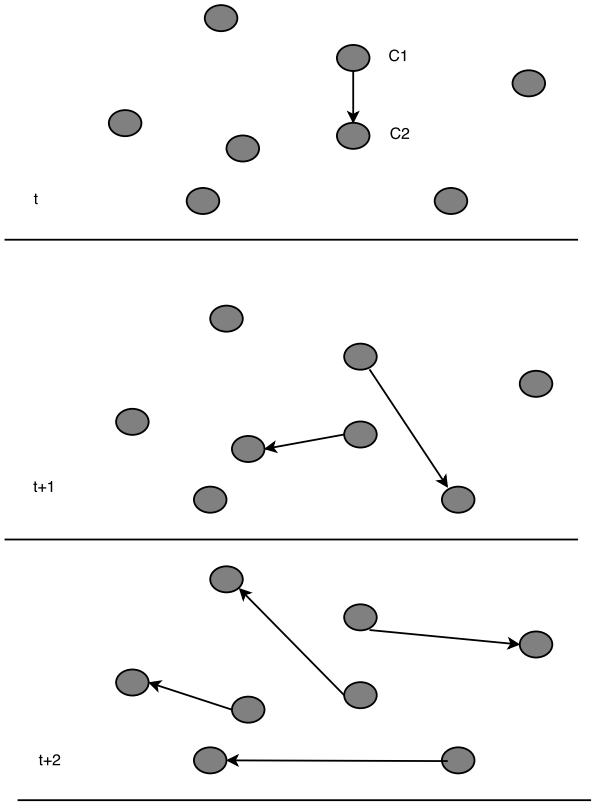


Figure 2.6: An Example of Information Dissemination using Gossip Protocol

and exploitative action is chosen otherwise. The value of the ϵ is also decreased per iteration. Initially, it is set to 1 to fully explore and learn. It is decreased at subsequent steps to exploit more than explore.

2.6.4 Gossip Protocol

Gossip protocol is an efficient way to globally disseminate information in a distributed system. In decentralized self-adaptive systems, dissemination of a single piece of information is required. For example, the calculation of global rewards needs local reward information from all the nodes. One node can spread this information in the entire system efficiently using the gossip protocol. This protocol is of two types which are *simple* and *aggregated gossip* [16].

Simple Gossip

Simple gossip protocol is shown in Figure 2.6. At time t, control loop $C1$ chooses a control loop $C2$ randomly and passes the information. At time $t+1$, $C1$ and $C2$ both passes the information to another randomly chosen node. This process continues at time $t+2$. Thus, gossip spreads information like epidemics [16]. When this transmission continues indefinitely, this is called *anti-entropy*. When it stops after a specific time, this is called *rumour mongering* [16]. Moreover, two kinds of policies can be followed when a control loop passes information to the other one. These are *push* and *push-pull policy*. In the first one, information transfers from sending to the receiving end. In the second one, information transfer occurs in both directions. Literature show that, in a gossip mechanism, a single information can propagate across the network in P_n rounds where P_n is given in Equation 2.3 [59].

$$P_n = log_2 n + lnn + O(1) \tag{2.3}$$

This indicates that information propagates very fast following this protocol.

Aggregated Gossip

In case of *aggregated gossip*, the simple protocol is followed with a single change. Each node computes the values of a function before disseminating the information. The functional value is disseminated. This can be useful in performing a collaborative computation. For example, consider finding the maximum number where each control loop contains a number. Firstly, a control loop passes its number n_1 to a randomly selected control loop. The control loop with number n_2 at the receiving end computes the value of the function $max(n_1, n_2)$ and passes it to another loop. Eventually, all the control loops agree on the maximum value.

Several gossip-based approaches have been proposed in the literature. Mostly use gossip for self-assembly where a set of services automatically find their depen-

dencies. The most prominent ones are present in [16], [60] and [21].

2.7 Summary

Self-adaptive system is a widely researched topic which has been explored from various areas. As a result, decentralization of control loop has been discussed from a variety of viewpoints such as rule-based, multiagent learning systems, biologically inspired systems etc. Rule-based systems use static rules and so, are applicable in small-scale systems. Biologically inspired approaches are driven by natural principles such as noise, emergence, evolution etc. Multiagent learning techniques are most commonly used in the literature to build decentralized self-adaptive systems. This is because it has a rich research background and directly goes along with this type of systems. This chapter provides the background knowledge required to understand the works in decentralized self-adaptive systems - their structure, advantages, weaknesses, applicability and research scope.

Chapter 3

Literature Review

Although there are multiple convenient techniques for centralized self-adaptation in the literature, very few approaches address decentralization. Among these approaches, most are generic frameworks and designs that provide the mechanism to build decentralized self-adaptive systems from a very abstract point of view. Some are rule-based approaches that define when and how the control loops coordinate by specifying hardwired rules [16, 21]. In multiple local and global goal optimization, the number of cases to consider becomes high. So, rule-based approaches become impractical. This is why learning-based approaches, specially reinforcement learning-based approaches are currently being suggested for this purpose.

According to the type of approaches adopted, the decentralized self-adaptation techniques can be divided into four classes which are mentioned below.

1. General Frameworks and Design
2. Rule-Based
3. Biologically Inspired
4. Learning-Based

The first two have already been discussed. The third one, the biologically inspired approach generally follows the collaboration mechanisms presented in the nature to model the coordination in decentralized self-adaptive systems [45, 61]. It considers each of the local control loops as a living organism and derives inspiration from nature for optimization and collaboration. Learning-Based approaches mainly use reinforcement learning where each local control loop, also called agent, receives a reward indicating how much it conforms to the local and global goals. The objective is to collaboratively select a strategy with the maximum reward considering other agents strategies. It is mentionable that both the learning-based and biologically inspired approaches are mainly related to the multiagent system domain. So, the classification is not necessarily orthogonal. In this chapter, the seminal approaches under these four classes are discussed. For every approach, the research problem, methodology, validation technique and research scopes are described.

3.1 Decentralized Approach to Self-Adaptation

The centralized approaches cannot be trivially extended to support decentralization as seen from the literature. This is why decentralization of control loops has emerged as a solution. In the previous section, the literature in the decentralization of control loops is partitioned into four classes. The works under these classes are presented in the following sections.

3.1.1 General Frameworks and Design

Frameworks and abstract designs are necessary to increase the understanding of a field which later inspires more concrete solutions. For this purpose, at the beginning of the decentralization of self-adaptive systems research, some high-level designs were proposed. IBM proposed the autonomic computing model that used

multiple control loops in a hierarchical fashion [11]. Tesauro et al. proposed an approach named Unity which supported multiple autonomic elements to collaborate in achieving self-assembly, self-healing and self-optimization [62]. Weyns et al. proposed a reference model for decentralized self-adaptive system where they separated the computation and coordination mechanism of the control loop [4]. Weyns et al. proposed five design patterns for decentralized self-adaptive systems to model the interactions among the MAPE-K control loops in each of the levels (monitor, analyze, plan and execute) [1]. These works are further explained below.

The IBM Autonomic Computing System

The autonomic architecture provided by IBM is one of the most popular one followed by numerous literature. The MAPE-K control loop has already been discussed in the Background chapter. This autonomic computing architecture contained two layers of autonomic managers that constituted a hierarchy [11]. Although the form of supported decentralization in the IBM autonomic computing architecture was limited, it is discussed here because it is considered an important initiative towards decentralized control [4].

The lowest level of the autonomic system was *managed resources*, for example, database servers, application servers etc. The allocation and control of managed resources were done using *touchpoints* [11]. These are interfaces for the controlling purpose. The touchpoints were managed by *touchpoint autonomic managers*. These performed self-adaptation by controlling resources through touchpoints. Another level of *Orchestrating Autonomic Managers* was used that controlled the autonomic managers of the lower level. These coordinated the lower level autonomic managers for system-wide goals. On top of these levels, a *manual manager* existed who was generally a human monitoring the whole system through a user interface.

The autonomic system by IBM is a hierarchical composition of multiple control

loops. However, due to scalability issues, fully decentralized solutions are more preferred [4]. The IBM autonomic system does not provide a solution towards full decentralization where the control loops act as peers in order to optimize the local and the global goals. Another problem of this approach is the high coordination overhead caused by all the orchestrating autonomic managers communicating with one another.

Unity Architecture

Autonomic computing enables management of a system in a more self-dependent way. Decentralized autonomous systems or self-adaptive systems closely resemble multiagent systems where multiple intelligent agents collaborate towards achieving particular goals. The concepts in multiagent system such as group formation, coordination, emergent global behavior from local ones are also present in the decentralized self-adaptation domain [62]. So, an approach towards designing decentralized self-adaptive systems inspired by multiagent systems can provide further directions toward this path. Tesauro et al. proposed the Unity architecture where multiple autonomous agents interacted to achieve self-adaptability [62].

The core element of the Unity architecture was the *autonomous element* [62]. Each component in this architecture was considered as an autonomous element. The duty of an autonomous element was to maintain its autonomic behavior both internally and externally. That is, it needed to perform self-adaptation for adapting itself and had to maintain external relationships for adapting the global system behavior [62]. Unity supported multiple *application managers* that managed multiple environments providing different application services. A *resource arbiter* element allocated resources to these application environments. It calculated and applied an overall optimal allocation for all sets of application environments. The *OSContainer element* helped to start services or any autonomic element. The *Registry element* helped to find the location of other elements for communication. The

Policy Repository element provided policy support for other elements. The *Sentinel element* assisted in monitoring. All the autonomic elements collaboratively worked for self-adaptation. For example, this paper proposed an approach named goal-driven self-assembly based on Unity architecture [62]. In this approach, an autonomic element searched the registry for resolving its dependencies. When found, it formed a relationship with the element it depends on. It registered itself to the registry when all of its dependencies were resolved. This paper also showed a utility-based approach to self-optimization. Application managers contained a utility function that provided utility given the service level and the demand of the user. Application managers tuned the internal parameters to get an optimal utility for a specific demand. Each application manager also calculated a maximum attainable value for each resource level and transferred these to the arbiter. It decided the most optimal resource allocation for all the application managers. Thus, self-optimization was performed using the Unity architecture.

Although Unity architecture provided useful research direction towards multiagent system based self-adaptation, it did not explicitly support peer to peer collaboration. The concept of satisfying global goals was centralized, for example, the arbiter helped to optimize the resource allocation centrally. Moreover, autonomic components were searched using a centralized registry. Overall, the approach can be made more scalable by incorporating a peer to peer decentralization scheme.

The Decentralized Self-Adaptation Reference Model

Centralized solutions to self-adaptation, as mentioned previously, did not explicitly support full decentralization. Only the MAPE-K loop supported hierarchical decentralization. The aforementioned Unity architecture also did not provide a way to build fully decentralized self-adaptive systems. Weyns et al. addressed this issue and proposed a model that enabled self-adaptation in a fully decentralized

manner [4]. This model contained an explicit *coordination model* that enabled control loops to collaborate in a fully decentralized way.

Weyns et al. discussed the technique in two parts namely meta-level computation and meta-level models [4]. The control loops consisted of four stages of the MAPE-K model which are monitor, analyze, plan and execute. Each stage performed some computations regarding the satisfaction of local goals and some coordination for the satisfaction of the global goals. The computation and coordination mechanism used four types of meta-level models which are *system*, *concern*, *working* and *coordination model*. First three are related to the computation where the last one is associated with coordination. *System models* were the models of the locally managed systems. *Concern models* contained the goals of the self-adaptive units (single control loop). *Working models* contained data structures and information that the stages of a control loop exchanged. *Coordination models* carried all the information regarding coordination among control loops. For example, a coordination model may contain the list of the peers to coordinate with. It may also contain the information regarding ongoing computations for coordination. Separating the coordination model displayed an explicit support of coordination among multiple control loops.

The model by Weyns et al. showed high interest in building fully decentralized self-adaptive systems satisfying local and global goals. However, they also addressed that the problem of partial knowledge and coordination overhead is difficult to solve. They mentioned that partial knowledge should be the norm as sharing complete knowledge among all the control loops is not scalable [4]. They also mentioned that coordination is needed but high exchange of messages may compromise the performance of the system. They left both of these as research challenges to be addressed in the future.

Patterns of Decentralized Control

Decentralized control occurs when there are multiple control loops and these require a coordination mechanism to collaboratively achieve a goal. Decentralized control can be realized in multiple ways. For example, multiple control loops can hierarchically collaborate where one control loop adapts another towards a high-level goal [1]. Again, multiple control loops can collaborate where one is the master loop that performs planning. Master delegates the monitoring and execution responsibility to the slaves [1]. To systematically identify and document these control loop patterns (recurring problems that have a common solution [63]), Weyns et al. discussed five patterns of decentralized control [1].

Weyns et al. identified five patterns for control loop interaction which are *Hierarchical Control*, *Master/Slave*, *Regional Planner*, *Fully Decentralized* and *Information Sharing* [1]. These patterns are described briefly next.

- *Hierarchical Control*: In this pattern, the control loops form a hierarchy. The lowest level control loop in this hierarchy manages the system level goals. As the level gets higher, control loops become more global in vision. These levels share information with one another to make decision. However, this layered architecture with hierarchically ordered goals can be complex to design.

- *Master/Slave*: In this pattern, a control loop becomes the master and performs the planning part. It delegates the task of monitoring, analyzing and executing to different slaves. However, in this pattern, the master becomes the central point of failure.

- *Regional Planner*: Multiple master/slaves are used to manage multiple regions of the system in this pattern. Some local hosts (similar to slaves) are connected to a regional planner (local master). The local host collects information, analyzes it and passes it to its regional planner. Each of the

regional planners in all the regions throughout the system coordinates to take a globally optimized decision.

- *Coordinated Control*: In this case, each control loop manages one subsystem and contains all the MAPE components. The monitor, analyze, plan and execute components of a control loop interact with corresponding components in the other control loops to perform adaptation. In this case, the rate of information transfer can be very high because all the components of the control loop participate in the interaction.

- *Information Sharing*: In this case, the structure of the fully decentralized system is maintained. However, only the monitor component takes part in interaction. This ensures that minimum information transfer takes place.

Weyns et al. mentioned that patterns that support fully decentralized solution (Coordinated Control and Information Sharing) are more scalable. They further mentioned that these two types of patterns need to be studied more thoroughly as the literature addressed only the other three. The paper also listed some challenges related to the patterns. One of the most important challenges was the incorporation of a knowledge pattern layer. Knowledge can be decentralized similar to the structural decentralization presented here in a separate layer. However, how local goals and global goals are satisfied in a fully decentralized knowledge setting cannot be answered directly from the pattern, rather needs to be studied separately.

The general frameworks and designs for decentralized self-adaptive systems addressed the research challenges in this domain and proposed abstract solutions to decentralized adaptation. However, concrete solutions are required both to increase understanding and to implement decentralized self-adaptive systems in the real life. The approaches presented in the other three categories describe such concrete approaches. These also do not acknowledge many addressed challenges,

mainly how partial knowledge can be effectively incorporated and coordination overhead can be reduced.

3.1.2 Rule-Based

Rule-Based approaches strictly follow a condition-action rule to specify when and how to coordinate and who to coordinate with. A less strict version of these chooses actions based on utility. In this case, agents choose to select an action that results in maximum utility. The utility functions are predefined and the agents strictly follow the utility function-based action selection without learning from the environment. As a result, even if the environment changes resulting in lower utility values, the agents are unable to sense it.

Self-Organizing Software Architecture for Distributed Systems

Architecture-based self-adaptation is quite popular in case of centralized self-adaptation techniques [64, 65, 66]. In architecture-based self-adaptation, the autonomic manager uses an architectural model to reason about the system and to perform reconfiguration. Inspired by the architectural approaches in centralized self-adaptive systems, Georgiadis et al. proposed an architecture-based self-organization design for distributed systems [67].

The design proposed by Georgiadis et al. maintained three runtime elements in each component [67]. One was the *component implementation*. It consisted of some ports with which the *component manager* can interact with. *Ports* also helped to bind the component to another component to either resolve its own dependency or resolve the dependency of the other. Events were generated when a binding took place and the component manager could listen to that event. Another element is the *component manager* which was responsible for maintaining the global view of components. It contained constraints which the component implementation had to follow during binding. When a component was removed

from the system or added to the system, a totally ordered broadcast message was sent to all the configuration managers. The self-organization was initiated by updating the configuration graph. It is mentionable that all the configuration changes occurred by achieving a lock over a change so that a consistent view could be maintained. Finally, binding and unbinding or self-organization took place. In this case, predefined condition-action rules were used by the component managers to bind or unbind components with a view to producing a fully resolved component. The graph was updated accordingly. While binding the components, architectural constraints are also maintained.

The approach was evaluated on an emulated platform in Java over a pipeline architectural constraint. It was observed that increasing pipeline resulted in a very quick increase in time to self-organize. So, a more efficient solution is required. In fact, this approach has multiple issues that need to be addressed. There is no concept of local or global goals. A totally ordered broadcast was performed to coordinate with all the components which is not scalable [16]. A global component model was used which contradicts the concept of partial knowledge for decentralized self-adaptive systems.

Flashmob

Self-assembly is an interesting problem in the self-adaptation domain. Here, a configuration of multiple components is given with some nonfunctional properties. The target is to derive a new configuration when the nonfunctional property values do not conform to nonfunctional goals. Previous approaches to the self-assembly problem such as [67] was not scalable due to a large number of message exchanges. So, Sykes et al. proposed a scalable mechanism to self-assembly using the gossip protocol [16].

In Flashmob, each node contained some components with interfaces. The state of a node was the global configuration that contained dependencies of all the com-

ponents. The dependencies were classified into two types which were *req* and *prov*. The first one specified that the component requires a specific interface and the second one specified that the component provides a particular interface. At first, all the nodes had local configurations. States were disseminated with gossip protocol (explained in Section 2.6.4 of the Background chapter). After receiving a state, it was considered whether one node can satisfy the dependency of the another by providing an interface. In this case, the dependency was added to the state. This resulted in a new state configuration with some of the dependencies resolved. However, if multiple components with the same type of interface could meet the dependency, the component with maximum utility value was chosen. Thus, a final configuration was derived when all nodes agreed on a solution. All nodes agreed when the solution was complete, that is, all dependencies were resolved. The solution also had to be valid, that is, system-wide structural constraints needed to be addressed. For this, every node computed whether the complete solution followed the structural constraints it knew. If any structural constraint failed, the self-configuration process was restarted from any previously stored state. Moreover, a death certificate was propagated to the nodes so that this solution could never be chosen again.

The Flashmob approach was evaluated using the *Koala* robots [68]. Each koala robot had to carry balls to a robotic arm. For this, it needed to search for the robotic arm following any of the three search patterns namely *Circular*, *ZigZag* and *Random*. Each search pattern had associated Quality of Service (QoS) values which could be used to calculate a utility. The robots consisted of multiple components in each of the Koala, Koala arm and location server nodes. Three essential components were selected initially and their dependencies were resolved following the mentioned approach. The technique was also applied in a large-scale simulation. It was seen that with the increasing number of nodes or the number of components, rounds to converge did not increase so rapidly. So, it achieved what

it promised. However, how to handle global QoS goals were not explicitly stated. Moreover, each state contained the complete state information which resulted in large coordination overhead.

QoS Aware Fully Decentralized Service Assembly

The previous approach, Flashmob, did not explicitly support the adaptation of global QoS goals. However, the adaptation of these type of goals is important and challenging for reasons mentioned previously. Moreover, the coordination overhead in Flashmob needs to be decreased, as it hampers the scalability and usefulness of the decentralized approach. So, Grassi et al. proposed a technique that solved these problems by restructuring the self-assembly problem and by defining some local and global utility functions [21].

Grassi et al. defined service by a five tuple of $(t, \delta, u, pred, succ)$ where t is the provided interface type, δ is the dependencies to be resolved, u is the local utility function, *pred* is the services that are bound to this service for resolving its dependency and *succ* is the services bound to this service for resolving their dependencies [21]. Local and global utility functions were defined which produced a higher value for goal conformance. The approach performed self-assembly in two threads. One thread actively disseminated information regarding $pred[S]$ and itself. Another thread received the disseminated information from another node. After receiving this information, it checked whether any of the services in it could solve one of its dependencies. In this case, the service was added to its *pred* set resolving one of its dependencies. However, if a similar type of service already existed in the *pred* set, it was checked whether the newly arrived service provided higher combined utility. Here, a combined utility was calculated by weighted average of all the local and global utilities in a node. In this case, the new service was added after removing the previous one. Thus, service assemblies were built without a complete state information in each node.

It was evaluated by PeerSim simulator where peer to peer connection of services was emulated [69]. It was seen that the approach quickly converged to an optimal solution. Even in case of failures, dropped utility values soon came back to optimal ones through reassembly. This approach solved the global QoS goal problem mentioned previously. However, the calculation of the utility function depended on the *pred* set. Each service required all the utility function values of the bounded services for calculation of its own utility. These utility values were transferred often which can result in a large number of message exchange. So, although it seems to reduce coordination overhead by reducing state information, the utility information was increased. Besides, it sent gossip messages to all the bounded services. Due to this, the number of messages also become large if the set of bounded services become large. So, partial knowledge incorporation and coordination overhead reduction need further attention.

On Interacting Control Loops in Self-Adaptive Systems

Self-adaptive systems have been used to solve many problems discussed in the Background chapter. One such problem is the coordinated traffic jam detection and traffic control. In this case, traffic jam is detected by some coordinating cameras. Initially, these cameras operate alone. These cooperate to form an organization to span their view where a master is in charge of the organization. One interesting self-adaptation scenario is the self-healing of these cameras. This paper discussed such scenario and proposed an extension to the MAPE-K loop suited for decentralized systems [44].

When a camera joins an organization, it may take the role of a master or a slave. Generally, the failure of a camera should influence its neighbours. However, in this paper, this depended on its role. For example, the failure of a master may influence its neighbouring nodes, slaves and the master of the neighbouring organizations. Again, the failure of a slave may affect the master and the neighbours. A self-

healing subsystem was added to each of the cameras. The self-healing controller provided the role of the camera to a MAPE-K manager which monitored the camera. It further monitors the self-healing subsystems of the cameras present in the dependency model (a set of other cameras that the camera depends on). In case of a failure, stored repair strategies were executed. These strategies worked on both *intra* and *inter-loop level* [44]. For example, if a master failed, the slaves, the other neighbours and the masters of neighbouring organizations stopped communicating with the master. The slaves chose another master and resumed working. It is visible that the execution of the MAPE-K loop cannot be done at once. Intra-loop coordination enabled computation within one MAPE-K loop to coordinate with one another. So, sub-loops were created within the MAPE-K loop [44]. Inter-loop coordination enabled coordination among multiple MAPE-K loops.

This approach showed how self-adaptation can be applied to the distributed traffic control domain. It also provided MAPE-K loop extensions to support for decentralization. However, due to using fixed rule-based approach, the selected actions will not be optimal in the long run. For example, choosing a specific master that has more failure rate may cause another failure scenario soon in future. So, learning-based approaches should be used to address these concerns.

Some of the aforementioned rule-based approaches addressed the partial knowledge and coordination overhead problems in the literature. However, none of these are fully decentralized from the knowledge perspective. Rule-based approaches are mainly applied in environments which change a little. However, previous chapters discuss that this rarely happens. In some self-organization problems, the environment may be simple enough for using a rule-based approach. This is why most of the rule-based approaches are for self-organization. Overall, the main problem of these approaches is that these do not learn from the environment. So, in the highly dynamic environment, optimal behavior cannot be expected.

3.1.3 Biologically Inspired Approach

A new approach towards self-adaptive systems is the biologically inspired approach. These approaches take inspiration from nature to solve a problem. Several biologically inspired approaches are present in different domains, for example, genetic algorithm in an optimization approach that follows biological evolution. In autonomic computing, biologically inspired centralized approaches exist for self-adaptation. For example, in [70], a self-adaptation approach similar to human autonomous nerve system has been proposed. However, biologically inspired decentralized control are recently being addressed following this line of research.

SelfLet

Self-organization is one of the self-adaptive behaviors where multiple agents collectively organize to form specific patterns. A biologically inspired approach may be compared to how human forms groups. A person may ask another person to introduce his known ones to him with some criteria, which is an active approach. Again, the other person may introduce his known ones having some criteria without being asked. This is the approach that this paper adopts. The SelfLet is, however, a framework followed by each agent that is capable of initiating self-organization according to the context [61].

This paper discussed the methodology in the context of two self-organization algorithms namely *clustering* and *reverse clustering*. In clustering, agents with similar types form groups by themselves. In reverse clustering, the opposite situation occurs. In the proposed clustering approach, a node acted as an initiator and connected two neighbours of similar types. After this, it disconnected from one of these two to keep the number of links constant. This continued until a specific homogeneity was achieved. Homogeneity was calculated using Equation 3.1.

$$H_c = \frac{\sum_i^k n(i)}{n_k} \tag{3.1}$$

Here, k is the total number of nodes, $n(i)$ is the number of distinct nodes of the same type and n_k is the total number of links. The execution was terminated after the homogeneity became low and did not change significantly in subsequent rounds. To perform reverse clustering, the nodes of different types were introduced keeping rest of the algorithm unchanged. Clustering is applicable to a load balancing scenario where reverse clustering is applicable when execution of a task dependent on another task [61]. In a decentralized load-balancing environment, for an example, an autonomic mechanism is needed to observe whether an agent has high load for delegating a task or a task dependency has arisen. This autonomic capability was provided by the SelfLet framework. It contained an *autonomic manager* for sensing the environment and for executing *adaptation rules*. Adaptation rules activated different *autonomic abilities* in specific context. For example, the self-organization algorithms can be included as autonomic abilities. Thus, the SelfLet approach merged different biologically self-organizing approach using an autonomic manager to create a self-adaptive system.

The approach was analyzed in three topologies namely random, torus and spiral [61]. It was seen that for both clustering and reverse clustering, random topology performed much better in terms of homogeneity. Moreover, the number of transferred messages did not increase significantly with node increase. However, the scenario of local and global goals was not addressed explicitly. Here, local goals and global goals are the same. Moreover, the termination of this algorithm required the calculation of homogeneity, which was performed centrally.

Clonal Plasticity

Clonal reproduction is a process by which a plant creates a copy of itself asexually. However, plasticity, which is the ability to adapt to the surrounding environment, is the reason for the difference between two copies [71]. The clonal plasticity, a concept proposed by Nallur et al., enabled agents to produce a variant by adapting

to its surrounding environment [45]. The clonal plasticity method acknowledged a decentralized environment with multiple loops. However, it assumed that global goals are achievable without considering coordination [45].

In this algorithm, each individual agent identified its *plasticity points* that could take multiple representations. For each plasticity point, the environment yielded rewards. If the reward was positive at time t and the previous reward was also positive at time $t-1$, the agent made an exact copy of itself. If the current reward was positive, but the previous reward was negative, it selected the previous plasticity point values. If the current reward was negative, new random plasticity values were calculated to generate a copy of the individual.

The approach was evaluated using the *Game of Life* [72]. In this game, a cell needs to adapt (dead or alive) according to its neighbouring cells. Each cell needs to change its neighbour calculation mechanism if environment requires. It was seen that the cells could individually reconfigure the neighbour calculation quickly as the environment indicated. However, this approach assumed that decentralized entities need not coordinate for a solution. Although some problems are by definition of this type, most of the real-life decentralized systems do not abide by this assumption [73].

Biologically inspired approaches are very recent and are still on development. These approaches have great potential because in Biology, coordination is a very common phenomenon from human group behavior to the animal world. So, biologically inspired approaches may provide the desired behavior for a decentralized self-adaptive system in the near future. The properties of a biologically inspired decentralized self-adaptation approach discussed in Section 2.6.2 of the Background chapter should be the main focus as research progresses.

3.1.4 Learning-based

Learning-based approaches assume that their operating environment is dynamic. Reinforcement learning-based approaches are the most used ones as these directly fit into the decentralized self-adaptation problem. In case of rule-based approach, actions are selected based on utility values. However, an optimal action in the current state may not be optimal when long-term consequence of the action is considered. Reinforcement learning helps to select actions considering the current utility values and the future utility values. Due to the dynamic characteristic of the environment, these total utility or reward values are updated continuously at every iteration. In the decentralized version, this update considers other agents' actions for both learning and action selection, as discussed in the Background chapter. The reinforcement learning-based approaches to decentralized control are discussed below.

Building Autonomic Systems using Collaborative Reinforcement Learning

In a decentralized environment, apart from its dynamic nature, two challenges occur for decentralized control. Firstly, to satisfy global goals, a global reward function is required to be known by all the agents. However, calculating global functions at a node requires complete state information which is not possible due to partial knowledge. Secondly, the agents communicate and transfer information to other agents. However, the knowledge of the neighbour about other neighbours can become invalid due to continuous change. So, a neighbour-level uncertainty also occurs. Collaborative reinforcement learning approach, presented by Dowling et al. addresses these cases [18].

In collaborative reinforcement learning, a global optimization problem was broken down into similar multiple discrete optimization problems that can be solved by each agent [18]. Each of the agents tried to solve the discrete optimization

problem using Q-learning. However, each of the neighbours also shared their optimal Q-value (cost) for solving the optimization problem. Every agent maintained its optimal Q-value and the neighbours' Q-values, whether neighbours' ones consisted of their shared Q-value and the connection cost. The lowest cost or the most optimal Q-value of solving the optimization problem was then observed. If the local agent could solve at this cost, actions were performed according to its Q-values. Otherwise, the discrete optimization problem was terminated at the agent and delegated to the lowest cost neighbour for further solving.

The technique was evaluated in an adaptive routing mechanism based on collaborative reinforcement learning named Sample [18]. Routing paths were adapted to control the congesting of the network, which results in overall higher throughput. It was compared with non-probabilistic routing protocols. It was seen that when system load was higher, routing was adjusted to achieve overall better throughput in the proposed approach. However, although the approach proposed to solve two significant problems of decentralized control, there is still a major drawback. It assumed that an optimization problem can be divided into multiple similar discrete optimization problems. However, this is difficult in many situations. For example, consider a response time threshold needs to be maintained for a global goal and different response time thresholds need to be maintained for local goals in different agents. The global goal cannot be easily divided into discrete optimization problems that each agent can solve.

Distributed W-Learning

Achieving optimal behavior in the lack of global knowledge is Challenging. The difficulty increases when multiple global goals are present. Each agent follows a policy for optimizing global goals. As agents are dependent, their policies become dependent too. Moreover, in a heterogeneous environment, where one agent state space does not match with another agent state space, the situation becomes

more challenging. Dusparic et al. proposed an approach where they extended *W-learning* proposed by Humphrys to decentralized context for addressing these challenges [33, 74].

A Distributed W-learning (DWL) agent contained a set of neighbours, a set of *local policies* and a set of *remote policies*. A Q-value was maintained for each state-action pair for each policy. A W-value was maintained for each state for each policy which indicated what happens when the policy is not followed. A higher *W-value* indicated that following the policy results in higher reward. For each set of local policies, Q and W-learning were initialized with local states and actions. In case of remote policies, these were initialized with remote states and local actions. Then, for each local policy, actions were selected according to the Q-values. Q-values and W-values were then updated. The similar was also performed for remote policy Q and W-values. Finally, after an iteration over all the policies, the action selection was performed following Equation 3.2.

$$W^* = max(W_{1L}, \cdot, W_{nL}, C \times W_{1R}, \cdot, C \times W_{nR}) \tag{3.2}$$

So, the maximum W value attached to a policy was chosen. Then, the action according to the highest Q-value of that policy was selected. Here, C is a cooperation coefficient which helps to give weights to the neighbouring agents' action preferences.

The technique was tested on an urban traffic simulator presenting a map of the Dublin's inner city [33, 75]. It consisted of 270 junctions where 62 were controlled by agents. Two policies were adapted namely optimizing global waiting time (GWO) and prioritizing public transport vehicles (PTO). Two metrics were used to test the performance which were average waiting time and the traffic density. It was seen that multiple policies performed better than single policies. It was also seen that the best result was achieved for the value of C between 0 and 1. The

distributed W-learning improves decentralized control loop design by coordination and multipolicy-based selection. However, the drawback was that W-values needed to be maintained for all the states for both remote and local agents. A simpler multipolicy-based technique is needed in this case. TThe technique also does not explicitly discuss what to do when local and global policies need to be followed corresponding to local and global goals respectively.

WSC-TMG Model

Wang et al. proposed a method for dynamic service composition that acted in a self-adaptive manner using multiagent reinforcement learning [19]. More specifically, they used a joint action learner [22]. They provided a mechanism to choose a unique solution every case when multiple optimal solutions are present, so that solutions are consistent. Their methodology is discussed below.

The system moves from one state to another when actions are applied to the agents. In their approach, an action was represented by choosing a service variant. For each state and joint action, a reward value was calculated that represented how much the state conformed to the individual goals after applying the joint action. The target was to maximize this reward at each agent. Q-learning provides the mechanism to achieve this. Every agent maintained a Q-value for each of the states and joint actions. After a state transition, total rewards were observed at each agent and Q-values were updated considering this current reward and maximum attainable Q-values from the subsequent states. So, the Q-value represents the maximum achievable reward value following a specific action in a specific state. The maximum Q-value in the subsequent steps for an agent depends on the action selection of the other agents as the Q-values are maintained for joint actions. So, each agent holds an opponent model, which is the frequency of action selections in a state by other agents. This is used to calculate the maximum expected Q-value from the subsequent states. An agent selects the action that maximizes its

expected Q-value for the current state. Here, an important thing to mention is that global reward values for a joint action are similar for all the agents. So, this corresponds to a coordination game where all the agents receive similar rewards. The opponent model approach is more formally called fictitious play approach, which ensures convergence to a unique optimal solution [53].

The self-adaptive service composition mechanism was evaluated using the Quality of Web Service (QWS) dataset [76]. It was observed that the proposed approach with Boltzmann exploration performed better than the greedy approach in terms of reward. Although it converged faster, increasing the number of services increased the convergence time. However, this model has much scope for improvement. Firstly, it considered the joint action of all the services which is not practical. This is due to the partial knowledge assumption in self-adaptive systems with decentralized control loops. The reward value calculation also assumed global knowledge. Besides, total rewards are calculated based on a fixed weighted summation of local reward values. However, sometimes these weights need to be dynamically updated. For example, consider four reward functions r_1, r_2, r_3, r_4 which returns greater than or equal to 0.5 on goal conformance and less than 0.5 otherwise. For reward values 0.7, 0.8, 0.3, 0.6, the total reward value is 0.6 with 0.25 as the weight. For reward values 0.6, 0.8, 0.5, 0.5, the total reward value is 0.6 too. However, the first reward value should be less than the second one. This is because in the first case, goal violation indicates the need for adaptation and so, the value 0.3 should be emphasized.

Reinforcement Learning Techniques for Decentralized Self-adaptive Service Assembly

The previous gossip driven approach by Grassi et al. was rule-based [21]. As a result, the facilities of the learning-based approaches were missing. Caporuscio et al extended the approach with learning capability [20]. They also incorporated

the MAPE-K loop into their decentralized self-adaptation approach.

The reward functions of the approach were defined similarly as [21]. A global QoS reward was defined that was the weighted summation of the all composite local QoS rewards. In the monitoring step, the agent (service) listened for gossip messages from the neighbours. A gossip message consisted of all the known services to the source service and the source service itself. Then, the gossip-based approach which was previously used in [21] to update the list of the known or bounded service was used. However, in the proposed technique, the known services were just the candidate ones which faced another level scanning in the analysis and planning phase. With the change of the known service list, plan triggered and the known services were checked in a similar way. However, learned reward values for a specific service at time t, similar to Q-values were used to compare and determine the service to bound to. Thus, a learned global reward value acknowledged the dynamic environment.

The technique was evaluated using PeerSim similarly as before. It was compared with other three approaches namely Random, Single-layer Reinforcement Learning (SRL) and greedy. It was seen that global quality values eventually became higher than the other approaches. Even when the peers left or joined the network, services reassembled to quickly derive an optimal solution. However, the weighting mechanism to derive a global reward function has the similar problem mentioned previously. Moreover, it was not explicitly mentioned whether joint action learners or individual learners are used. The technique to calculate global reward functions required dissemination of each local reward function values to each agent. How this can be done efficiently was not mentioned.

Learning-based approaches provide a convenient way to achieve decentralized self-adaptation. The reinforcement learning, specially multiagent reinforcement learning directly fits into the decentralization of control loop problem. However, as the discussed papers indicate, high level of partial knowledge incorporation and

coordination overhead reduction are hard to achieve in real life systems.

3.2 Summary

Decentralization of control loop is still an important challenge for self-adaptive systems. Decentralization of control loop for self-adaptive systems can increase scalability and make self-adaptive systems more fault tolerant. Reinforcement learning-based approaches are more popular for this purpose. Some techniques to design decentralized self-adaptive systems have been proposed. None of these techniques address partial knowledge and coordination overhead together, which provides the scope for a major contribution in this domain.

Chapter 4

Decentralized Self-Adaptation with Partial Knowledge and Reduced Coordination Overhead

All the control loops need to communicate their strategies to one another for coordination in decentralized self-adaptive systems. As discussed in the previous chapter, partial knowledge should be the norm rather than this type of holistic knowledge. Decentralized self-adaptation techniques in the literature use multiagent Q-learning which requires this type of coordination. In the multiagent system literature, Interaction Driven Markov Game has been proposed which segregates coordinating states from the whole state space [5]. Coordination occurs in these states where only a single agent is considered. As a result, both partial knowledge and coordination overhead are taken into account. The decentralized self-adaptation problem has been fit into the Interaction Driven Markov Game and an algorithm to learn the coordinating states has been proposed. Reward functions have been appropriately designed for this purpose. To support multiobjective self-adaptive systems, a weighted sum of all the reward functions are used. A dynamic weight update mechanism is also proposed for better goal conformance

at runtime.

4.1 Problem Formulation

In this section, the problem addressed in this thesis is formalized. At first, some definitions used in this context are given below.

Definition 4.1. A **state** is a combination of metric values. If the metric values are continuous, the combinations of encoded metric values are considered as states. Here, encoded metric values are the ones that are classified into value ranges, for example, 0-2 seconds response time as low, 2-4 seconds as medium and >4 seconds as high etc.

A state can be defined either for a single or multiple locally managed systems. When a state is defined combining multiple locally managed system states, it is called the *joint state*. If $s_1, s_2, \cdots, s_n$ are the set of states of n locally managed systems, $S = s_1 \times s_2 \times \cdots \times s_n$ is the set of all possible joint states.

Definition 4.2. An **action** is a variant selection by each of the locally managed systems. To say differently, an action is an option at hand that, when applied, results in altered goal values.

Joint action can be defined similarly as joint states for multiple locally managed systems. If $a_1, a_2, \cdots, a_n$ are the set of actions of n locally managed systems, the set of all possible joint actions is $A = a_1 \times a_2 \times \cdots \times a_n$. For example, each Tele Assistance System(TAS) service (locally managed system) from the Introduction chapter calculates the state $s_i = \{fr_i, t_i, v_i\}$ where fr_i, t_i and v_i are the failure rate, response time and cost of the service in the i^{th} time instant respectively. For two services s_1 and s_2, their joint state can be written as $S_i = \{\{fr_i^1, t_i^1, v_i^1\}, \{fr_i^2, t_i^2, v_i^2\}\}$. Additionally, the action set of the Medical

Table 4.1: Goal Specification of Tele Assistance System

Goal	**Metric**	**Goal Type (Scope)**	**Goal Type (Target)**
Each service failure rate must be less than a specific local threshold	Failure Rate	Local	Threshold-based
Each service must have response time less than a local threshold	Average Response Time	Local	Threshold-based
At least one service must have failure rate less than a specific threshold	Failure Rate	Global	Threshold-based
Average cost per service must be minimized	Average Cost Per Service	Global	Minimization

Analysis Service is {*MedicalAnalysisService1, MedicalAnalysisService2, MedicalAnalysisService3*}. In case of the adaptive robot navigation example, the position of a robot in the Cartesian coordinate is considered as its state. The available actions are left, right, top and bottom, considering that the action will not lead to an invalid state, for example, hitting the wall or going outside of the grid. In a specific state, decentralized self-adaptation helps to execute the action with the maximum goal conformance.

Definition 4.3. The function that measures goal conformance is called the reward function. Generally, better goal conformance leads to higher reward.

In this thesis, reward functions have been defined based on the type of goal. Two classifications of goals are derived analyzing the literature which are scope and target based [4, 55, 77]. Scope-based goals are divided into local and global goals. Considering type, goals are divided into threshold-based, minimization and maximization goals. Threshold-based goals aim to keep the metric values under a specific threshold (Table 4.1). Minimization and maximization goals focus on minimizing and maximizing a specific metric value respectively (Table 4.1). The reward function for threshold-based goals has the form $R : G \times T \rightarrow [0, 1]$, where G is the set of current metric values related to a specific goal and T is a singleton containing the threshold. The minimization and maximization reward functions

can be written as $R : G \rightarrow [0, 1]$.

Table 4.1 shows the goal types associated with TAS. Here, the first three goals are clearly threshold-based ones. The last one is a global minimization goal as it targets minimizing the system-wide average cost per service. The local goal of the adaptive robot navigation problem is to find the shortest route to goal. So, if the distance between the robot's coordinate and the goal coordinate is taken as the metric, it becomes a minimization problem with local scope.

Formally, for each i^{th} locally managed system, in a specific joint state S, a joint action A must be selected such that $S \times A \rightarrow R_i$ is maximum, where R_i is the reward function. If there are multiple reward functions, that is, for multi-objective systems, R_i represents the composite reward function. This is defined as the weighted summation of all the reward functions.

$$R_{sum} = w_1 \times R_1 + w_1 \times R_2 + \cdots + w_n \times R_n, \sum_{i=1}^{n} w_i = 1 \tag{4.1}$$

Now, the following observations are made.

1. If there are n_e encoded metric values, n_m metrics and n_a actions for each n_s locally managed system, the total number of joint state, action combination is $n_e \times n_m \times n_a \times n_s^2$. As mentioned previously, a reward value needs to be maintained for each of these joint state-action combinations in each locally managed system. As multiple control loops must coordinate to calculate this reward, large coordination overhead occurs throughout the large joint state space.

2. It was observed that in some states only local goals were violated but global goals were satisfied. As only local goals are violated, decisions can be taken individually in these states. It means that no coordination is required in these states.

Table 4.2: Mathematical Symbols Used in This Chapter

Symbol	**Definition**
S	The set of all possible joint states
S_i	The set of individual states of the i^{th} agent
S^I	The set of coordinating states defined in the interaction game in IDMG
A	The set of all possible joint actions
A_i	The set of individual actions of the i^{th} agent
a_i	A specific action of the i^{th} agent
a_i^I	A specific action of the i^{th} agent in the interaction game
a_{-i}^I	A specific action of the other agent except the i^{th} agent in the interaction game
R	Any reward function
R_i	The reward function of the i^{th} agent
R^I	The interaction game reward function in IDMG
r_i	A specific reward value of the i^{th} agent
R_θ	The threshold-based reward function
$R_<$	The minimization reward function
$R_>$	The maximization reward function
R_{sum}	The composite reward function
r_i^{tot}	The total reward value of the i^{th} agent following an IDMG
r_c	The predefined coordination reward
G	The set of current goal values for an agent
m	A specific current goal value for an agent
m_{max}	The maximum observed value of m for an agent
T	A singletone of a specific goal threshold
θ	A specific goal threshold
th	The goal threshold for a violated goal requiring weight update
λ	The transition function of the interaction game defined in IDMG
λ_i	The transition function of the MDP defined in IDMG
$Q_i^*(s,a)$	The optimal overall Q-function
$Q_i^*(s_i,a_i)$	The optimal MDP Q-function
$Q_I^*(s,a)$	The optimal interaction game Q-function
α	The learning rate from the Q-value update
γ	The discount factor from the Q-value update
ϵ	The variable that controls exploration-exploitation ratio
n_r	The total number of reward functions
n_a	The total number of reward functions that requires weight update
fr	The fraction of the initial reward that is used during weight update
Δw_i	The amount to add to the violated goal weights for weight update

3. In the multi-objective scenario, the violated goals are the most important ones in a specific state. These should be assigned more weights during adaptation. So, the weights need to be updated dynamically.

In the next section, the proposed approach is summarized. The mathematical symbols used in this chapter are presented in Table 4.2.

4.2 Overview of the Approach

This section contains the core parts of the technique which are later described in details throughout this chapter. The following points are the main constitutes of the proposed approach.

- The proposed approach is based on the Interaction-Driven Markov Game (IDMG) [5]. IDMG helps to separate states where coordination is required between peers and where individual decision making is adequate. This is different from the traditional Markov Game which performs coordination in every joint state. As peer to peer coordination is used rather than a full joint coordination, partial knowledge is addressed. As coordination happens in some specific states only, coordination overhead is also reduced.

- An IDMG-based Q-learning technique is proposed to learn where coordination is needed in partial knowledge setting. In this technique, two types of Q-values are maintained which are individual and peer Q-values. An action called σ action is added to the individual Q-value action set. This action helps to detect where coordination is necessary based on negative rewards or penalties. As global goals require coordination, penalties are issued only when these goals are violated. In this case, a large coordination reward value is associated with the σ action to mark the corresponding state as a coordinating state. The peer Q-values, which are maintained for joint peer states and actions, are used in these coordinating states for action selection. Otherwise, the individual Q-values are utilized.

- An assumption of this approach is the homogeneity of the agents. Due to this, any agent can be called a peer. That is, the agent selection can be dynamic. This is important because an agent may not depend on the same agent in every state. It is mentionable that the agent selection technique is problem specific.

- Reward functions are defined according to the goal types. The reward weights are adjusted dynamically with $\Delta w_i = k \times (th - r_i)$. Here, th is the goal threshold, r_i is the current metric value and k is determined according to Lemma 4.1.

4.3 Interaction Driven Markov Games (IDMG)

An IDMG is defined as a tuple (M_1, M_2, I) where M_1 and M_2 are Markov Decision Processes (MDP) of the two locally managed system peers and I is an interaction game [5]. Each MDP $M_i = (S_i, A_i, R_i, \lambda_i)$ is defined as follows,

- S_i is the set of individual states of a locally managed system.
- A_i is the set of actions of the i^{th} locally managed system.
- R_i is the reward function of the i^{th} locally managed system.
- λ_i is the transition function defined by $S_i \times A_i \rightarrow u(s_i)$. Here, $u(s_i)$ is a function that assigns probability to the states.

So, M_1 and M_2 are basically Markov games when joint states and actions are replaced by individual states and actions, that is, coordination is not considered. The interaction game $I = (2, S^I, (A_1, A_2), R^I, \lambda)$ is a Markov game with both of the peers defined in the previous section. This is defined as follows.

- 2 is the total number of locally managed systems participating in the interaction.
- $S^I \subset S_1 \times S_2$ is the set of joint states requiring coordination.
- (A_1, A_2) is the set of joint actions of these locally managed systems.
- R^I is the reward function of the interaction game.
- λ is the transition function of the interaction game.

Here, In the coordinating states of the interaction game, Q-values are maintained considering the joint states and actions. In other states, individual states and actions are used similar to the aforementioned MDPs.

The total reward for a specific agent in IDMG is calculated using Equation 4.2.

$$r_i^{tot} = r_i(s_i, a_i, s_i') + r^I(s, a, s') \tag{4.2}$$

In coordinating states, that is, in states $s' \subset S^I$, $r^I(s, a, s') \neq 0$.

Spaan et al. showed that the optimal Q function $Q_i^*(s, a)$ for an IDMG agent can be approximated as follows [5].

$$Q_i^*(s, a) \approx Q_i^*(s_i, a_i) + Q_I^*(s, a) \tag{4.3}$$

Equation 4.3 shows that the optimal Q function is the summation of the optimal MDP Q-function $Q_i^*(s_i, a_i)$ and interaction game Q function. $Q_I^*(s, a) = 0$ in states where $s' \not\subset S^I$, that is, when $r^I(s, a, s') = 0$. In these states, optimal Q values are approximated by individual Q-learning. Otherwise, the joint Q-learning is used with two agents as peers to approximate the optimal solution.

Example

Consider the peer of the Medical Analysis Service as the Alarm Service. The IDMG for Medical Analysis Service of TAS can be defined as (M_m, M_a, I), where M_m and M_a are the MDPs of Medical Analysis Service and its initial peer Alarm Service. I is the interaction game in coordinating states. Let, $s^I = \{\{\{$*high,low,low*$\}$, $\{$*high,low,high*$\}\}\}$ is the only coordinating state for both of the services. Also, let, $A_m = \{\{$*MedicalAnalysisService1, MedicalAnalysisService2, MedicalAnalysisService3*$\}\}$ for Medical Analysis Service and $A_a = \{\{$*AlarmService1, AlarmService2, AlarmService3*$\}\}$ for Alarm Service. So, $M_m = (S_i - \{$*high, low, low*$\}, A_m, R_i, \lambda_i)$ and $M_a = (S_i - \{$*high, low, high*$\}, A_a, R_i, \lambda_i)$. The interaction game for Medical

Analysis Service is $I = (2, s^I, (A_m, A_a), R^I, \lambda)$.

4.4 IDMG-Based Decentralized Self-Adaptive Systems

The proposed approach is an IDMG-based decentralized self-adaptation technique that uses a single agent as a peer to decide jointly in coordinating states and independently in non-coordinating states. In this section, the proposed technique is described in detail. At first, the reward function definitions are provided. Then, the learning of coordinating states are described followed by a detailed example of this technique.

4.4.1 Defining Reward Functions

In this paper, three types of reward functions are considered for the three types of aforementioned goals. These reward functions are mentioned in Equation 4.4, 4.5 and 4.6.

$$R_\theta = \begin{cases} \dfrac{2 \times \theta - m}{2 \times \theta} & \text{if } \theta - m > 0 \\ \dfrac{1}{2} - \dfrac{\theta - m + 1}{2 \times (\theta - m_{max} + 1)} & \text{if } \theta - m < 0 \\ \dfrac{1}{2} & \text{if } \theta - m = 0 \end{cases} \tag{4.4}$$

$$R_< = \frac{1}{m + 1} \tag{4.5}$$

$$R_> = \frac{m - m_{max}}{m_{max} - m_{min}} \tag{4.6}$$

Equation 4.4 is for threshold-based goals where θ is the threshold. Observe that this reward function also provides an output maintaining a threshold of 0.5. That is, a less than 0.5 value is generated if the goal is violated and the opposite otherwise. This is important because it helps retain the property of easy

goal violation detection from threshold. This reward function is further explained below.

- The first part provides a value $(0.5, 1]$. The lowest value of the metrics is 0 (a non-negative metric function is assumed). Setting this value to the first part produces 1 which is the highest value of this function. The minimum value is calculated below.

$$\begin{aligned}\lim_{m \to \theta} \frac{2 \times \theta - m}{2 \times \theta} &= \frac{2 \times \theta - \theta}{2 \times \theta} \\ &= \frac{\theta}{2 \times \theta} \\ &= 0.5 \end{aligned} \tag{4.7}$$

- The value range of the second function is $[0, 0.5)$. The highest value proof is similar to Equation 4.7. The minimum value is calculated as follows.

$$\begin{aligned}\lim_{m \to m_{max}} \frac{1}{2} - \frac{\theta - m + 1}{2 \times (\theta - m_{max} + 1)} &= \frac{1}{2} - \frac{\theta - m_{max} + 1}{2 \times (\theta - m_{max} + 1)} \\ &= 0.5 - 0.5 \\ &= 0 \end{aligned} \tag{4.8}$$

The minimization and maximization reward functions are defined by Equation 4.5 and 4.6 respectively. Here, m is the current value of the metric.

These equations are directly used for local goals. However, as discussed earlier, global goals should return a non-zero value in coordinating states. At first, the coordinating state is defined below.

Definition 4.1. The state where any global goal is violated is called a coordinating state.

As global goals require coordination, the rationale behind this definition is intuitive. According to the discussion in Section 4.3, global reward functions,

as a part of the interaction game, should return a non-zero value in coordinating states and zero otherwise. Nevertheless, although the violation of global threshold-based goal is easy to detect, identifying the global minimization and maximization goal violations is harder. These two types of goals are partitioned into smaller local optimization problems. For example, the fourth goal from Table 4.1 can be partitioned into the problem where each of the services will attempt to minimize its own cost locally. This is represented by Equation 4.9.

$$R^i_< = \frac{1}{m_i + 1} \tag{4.9}$$

So, rather than minimizing the average total cost of all the services, each i^{th} service considers minimizing their own cost m_i. This is a local reward function which can be handled similarly to the other local reward functions.

The threshold-based global reward functions are designed considering it as the reward function of an interaction game. To support this, Equation 4.4 is modified as follows.

$$R^I_\theta = \begin{cases} 0 & \text{if } 0.5 - R_\theta < 0 \\ -R_\theta & \text{if } 0.5 - R_\theta \geq 0 \end{cases} \tag{4.10}$$

The equation returns a negative value if a goal violation occurs. This can be considered as a miscoordination penalty [7]. It helps to discourage the selection of action that leads to goal violations.

The weights in Equation 4.1 are updated at runtime for providing more emphasis on reward functions that indicate goal violation in a specific state. For example, consider four reward functions r_1, r_2, r_3 and r_4 which returns greater than or equal to 0.5 on goal conformance and less than 0.5 otherwise. For reward values 0.7, 0.8, 0.3 and 0.6, the total reward according to Equation 4.1 is 0.6 (with an equal initial weight of 0.25). For reward values 0.6, 0.8, 0.5 and 0.5, the total reward is also 0.6. However, the first total reward value should be less than the second

one because in the first case, goal violation indicates the need for adaptation and so, the value 0.3 demands more attention. So, the weight for 0.3 can be updated to 0.3542 from 0.25 and the remaining weight 0.6458 can be equally distributed among the other three agents. In this case, the total reward value becomes 0.558 which is lower than 0.6.

Lemma 4.1. Let n_r, n_a be the total number of reward functions and the number of reward functions requiring weight update respectively. Let r_i, th *and* fr be the i^{th} reward function value requiring weight update, reward function threshold and the fraction of the initial reward used to update the weight respectively. The weight of the i^{th} reward function is updated with $k = \frac{1}{fr \times n_r \times (n_a \times th - \sum_{i=1}^{n_a} r_i))}$ where $\Delta w_i = k \times (th - r_i)$.

Proof. $\Delta w_i = k \times (th - r_i)$

According to the definition of fr,

$$\begin{aligned}
\frac{1}{fr \times n_r} &= \sum_{i=1}^{n_a} \Delta w_i \\
&= \sum_{i=1}^{n_a} k \times (th - r_i) \\
&= n_a \times k \times th - k \sum_{i=1}^{n_a} r_i \\
&= k(n_a \times th - \sum_{i=1}^{n_a} r_i)
\end{aligned}$$

$$\therefore k = \frac{1}{fr \times n_r \times (n_a \times th - \sum_{i=1}^{n_a} r_i))} \tag{4.11}$$

□

Example

The example from the previous section is calculated here. For reward values 0.7, 0.8, 0.3 and 0.6, the composite reward value is given below.

$$\begin{aligned} r_i^{tot} &= 0.25 \times 0.7 + 0.25 \times 0.8 + 0.25 \times 0.3 + 0.25 \times 0.6 \\ &= 0.6 \end{aligned}$$

During weight update, if fr is chosen as 2.4, k is calculated as follows.

$$\begin{aligned} k &= \frac{1}{fr \times n_r \times (n_a \times th - \sum_{i=1}^{n_a} r_i))} \\ &= \frac{1}{2.4 \times 4 \times (1 \times 0.5 - 0.3)} \\ &= 0.52 \end{aligned}$$

Using this, the weight of 0.3 is updated using Δw_i.

$$\begin{aligned} \Delta w_i &= k \times (th - r_i) \\ &= 0.52 \times (0.5 - 0.3) \\ &= 0.104 \end{aligned}$$

So, the weight of 0.3 becomes $0.25 + 0.104 = 0.354$. The rest of the total weight $1 - 0.354 = 0.646$ is equally divided among the other three rewards. In this case, the composite reward value becomes as follows.

$$\begin{aligned} r_i^{tot} &= 0.2153 \times 0.7 + 0.2153 \times 0.8 + 0.354 \times 0.3 + 0.2153 \times 0.6 \\ &= 0.15071 + 0.17224 + 0.1062 + 0.12918 \\ &= 0.558 \end{aligned}$$

Algorithm 4.1 Algorithm for IDMG-Based Q-learning

1: $a_i = a_i \cup \{\sigma\}$
2: $Q_i(s_i, a_i) \leftarrow 0, \forall s_i, a_i$
3: $Q_i^I(s, a) \leftarrow 0, \forall s, a$
4: $r_c = C$
5: **while** $TerminationCondition \neq true$ **do**
6: $a_i \leftarrow selectAction(Q_i, s_i)$
7: **if** $a_i = \sigma$ **then**
8: $a_i^I = selectAction(Q_i^I, s)$
9: $a_{-i}^I \leftarrow receivePeerAction()$
10: $a \leftarrow a_i^I \cup a_{-i}^I$
11: *observe state transition* s_i' *and reward* r_i
12: $Q_i^I(s, a) \leftarrow Q_i^I(s, a) + \alpha \times [r_i(s, a) + \gamma \max_{b_i \in a_i \setminus \{\sigma\}} Q_i(s_i', b_i) - Q_i^I(s, a)]$
13: **if** $r_i < 0$ **then**
14: $s_i.isCoordinateState = true$
15: $Q_i(s_i, a_i) \leftarrow Q_i(s_i, a_i) + \alpha \times [r_c + \gamma \max_{b_i \in a_i \setminus \{\sigma\}} Q_i(s_i', b_i) - Q_i(s_i, a_i)]$
16: **end if**
17: **else**
18: *observe state transition* s_i' *and reward* r_i
19: **end if**
20: **if** $s_i.isCoordinateState = false$ **then**
21: $Q_i(s_i, a_i) \leftarrow Q_i(s_i, a_i) + \alpha \times [r_i(s_i, a_i) + \gamma \max_{b_i \in a_i \setminus \{\sigma\}} Q_i(s_i', b_i) - Q_i(s_i, a_i)]$
22: **end if**
23: **end while**

4.4.2 Learning to Coordinate

In the previous sections, it has been discussed that IDMG-Based Q-learning can be successfully used to design a decentralized self-adaptive system considering both partial knowledge and coordination overhead. However, two questions need to be answered for designing such a solution. Firstly, how are the coordinating states learned? The standard IDMG includes the coordinating state in the interaction game by definition [5, 7]. It is necessary to learn the coordinating states rather than prespecifying it due to scalability purposes. Secondly, how this IDMG-Based Q-learning supports peer to peer learning? Considering single agent to coordinate with, is important to support partial knowledge assumption. In this section, the IDMG-Based Q-learning is described where the learning of coordinating states are explained. Then, the second question is answered.

Algorithm 4.2 Algorithm for selectAction Function

Require: Q and s

1: $\epsilon \leftarrow k$
2: $p_r \leftarrow random(0, 1)$
3: **if** $p_r < (1 - \epsilon)$ **then**
4: **return** $\arg\max_{a_i} Q(s, a_i)$
5: **else**
6: **return** $a_i \leftarrow randomSelect()$
7: **end if**

Algorithm 4.1 shows the IDMG-Based Q-Learning for a locally managed system [7]. At first, a pseudo-action σ is added to the available action set. This action helps to learn the coordinating states. Two types of Q-values are maintained which are Q_i (individual Q-value) and Q_i^I (joint Q-value). The first one is for the states without coordination and the second one is for the coordinating states or for the interaction game. A reward called the coordination reward is initialized to a fixed large value (line 4). At first, an action is selected based on Q_i (line 6). If this action is the pseudo-action, it is considered as a coordinating state (line 7). In this state, an action is selected based on the value of Q_i^I (line 8). As this is considered as a coordinating state, the action and the state of the peer are perceived and $Q_i^I(s, a)$ is updated using the joint action and state (line 10-12). In the Q_i^I update at line 12, two observations can be made. Firstly, rather than choosing the next action that maximizes Q_i^I, the next action that maximizes Q_i is used. This is for maintaining the sparseness of Q_i^I [7]. Secondly, observe that $r_i(s, a)$ is calculated as the interaction game reward. So, when $r^I = 0$, the Q-value update becomes similar to the individual Q-value update at line 20. So, Q_i^I is updated with the miscoordination penalty only at coordinating states, which means that coordinating states are learned properly. In line 13 and 14, if a penalty is received, the state is marked as a coordinating state. This is because an action in this state may lead to a transition to the state where global goals are violated. So, coordination is necessary in this state to refrain from choosing this action. The individual Q-value in this state for σ action is updated with the large coordination

reward r_c (line 15). This will result in large Q-value for this state and σ action pair, which will help to always choose this action, and so, to coordinate in this state. Thus, coordination is chosen where it is necessary and skipped otherwise. However, if the selection action is not a pseudo-action the reward and next state is observed and the local control loop updates the Q_i individual Q-value (line 18-21). It is notable that if the coordinating state is already identified, updating the individual Q-value for the σ action is obviously not required (line 21).

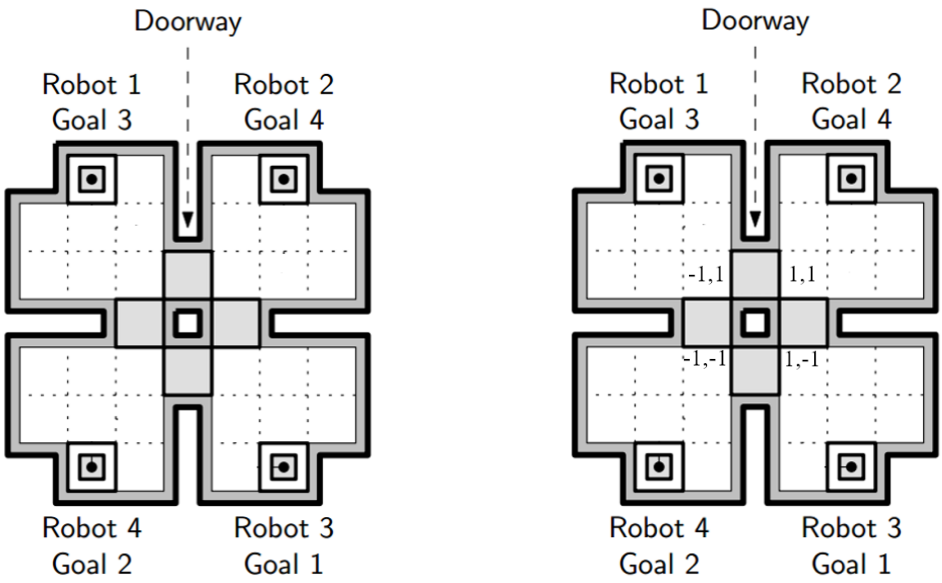
Algorithm 4.2 shows the *selectAction* function. For action selection, the $\epsilon - greedy$ strategy is followed because it is a simple but effective technique for multiagent Q-learning [22, 78]. An ϵ value is prespecified and actions are selected greedily according to the Q-values with probability $1 - \epsilon$ (line 1-4). Otherwise, a randomly selected action is returned. This helps to explore rather than only to exploit, which may result in finding a more optimal action [22].

An intuitive explanation of how Algorithm 4.1 learns the optimal strategy is given below.

- Firstly, it is important to show that all the coordinating states are found by this algorithm. If an appropriate exploration technique is chosen, all the state-action pairs are visited infinitely often. As a result, penalties will eventually be produced from all the coordinating states leading to their discovery.

- In non-coordinating states, global goals are satisfied. So, only local reward functions used to perform individual Q-learning. This is guaranteed to converge to the optimal solution according to the literature [58].

- In coordinating states, the peer Q-values are used. Although it seems that using single agent instead of multiple agents harm convergence, the single agent usage approximates the multiagent-based policy. This is because of the assumption of homogeneity of agents. As the algorithm runs infinitely



(a) Adaptive Robot Navigation inside a Grid

(b) Representation of the Grid in Cartesian Coordinate System

Figure 4.1: Example of Adaptive Robot Navigation inside a Grid

often, each agent selects a peer agent at every step infinitely often. The corresponding Q-value is updated according to the strategy of this peer agent. Each agent infinitely cooperates with each agent and so, it approximates the expected value of the strategies of all the other agents. As a result, it becomes an approximation of the fictitious play approach described in Section 2.6.3 of the Background Study chapter. As fictitious play-based multiagent Q-learning converges to the nash-equilibrium policy, the proposed technique also converges [79].

4.4.3 Example

In this section, the proposed technique is explained through an example. The toy problem of adaptive robot navigation is used. This is because it is much simpler to explain and visualize than TAS. The adaptive robot navigation problem from the Introduction chapter is shown in Figure 4.1. The learning steps of this problem are presented in the following subsections. The testing phase and the corresponding

Table 4.3: Initial Setup for Adaptive Robot Navigation

	Robot 1	**Robot 2**	**Robot 3**	**Robot 4**
Initial State	-2,3	2,3	2,-3	-2,-3
Goal State	2,-3	-2,-3	-2,3	2,3
Learning Rate	0.84	0.525	0.525	0.95
Decay Factor	3.4	6.7	6.7	6.7

steps are given in Appendix A.

Reward Function Selection

The local goal of the robots is to reach their goals in the shortest paths. The global goal is to avoid conflict. The local goal is a minimization goal as the difference between the goal state and the current state needs to be minimized. So, it is defined as follows.

$$R^i_< = \frac{1}{d_i + 1} \tag{4.12}$$

Here, d_i is the Manhattan Distance. Manhattan Distance is the summation of horizontal and vertical distance between two points [80]. The global reward function can be calculated as a threshold-based goal using Equation 4.4. However, as it is a single-objective problem, a static penalty of 20 is given when conflict occurs. Reaching goal state yields an additional reward of 10.

The initial setup is given in Table 4.3. It is mentionable that, a fixed learning rate and decay factor are used for demonstration purpose. Here, $\epsilon = \frac{1}{decay\ factor}$.

Learning to Coordinate

Step 1 - Robot 1

According to epsilon greedy strategy, Robot 1 chooses actions greedily with probability $0.706 = (1 - \frac{1}{3.4})$ and randomly with probability $0.294 = \frac{1}{3.4}$.

At first, the individual Q-table is empty. So, Robot 1 makes a random move from 2 of the valid moves (right and down). Here, the random move is chosen as down.

Individual Q-table for Robot 1 is updated as follows.

$$\begin{aligned} Q((-2,3),down) &= Q((-2,3),down) + \alpha \times (reward + \gamma \times \max_b(Q((-2,2),b))) \\ &= 0 + 0.84 \times (\frac{1}{9+1} + 0.7 \times 0) \\ &= 0.084 \end{aligned}$$

Step 1 - Robot 2

According to epsilon greedy strategy, Robot 2 chooses actions greedily with probability $0.851 = (1 - \frac{1}{6.7})$ and randomly with probability $0.149 = \frac{1}{6.7}$.

At first, the individual Q-table is empty. So, Robot 2 makes a random move from 2 of the valid moves (left and down). Here, the random move is chosen as down.

Individual Q-table for Robot 2 is updated as follows.

$$\begin{aligned} Q((2,3),down) &= Q((2,3),down) + \alpha \times (reward + \gamma \times \max_b(Q((2,2),b))) \\ &= 0 + 0.525 \times (\frac{1}{9+1} + 0.7 \times 0) \\ &= 0.0525 \end{aligned}$$

Step 1 - Robot 3

According to epsilon greedy strategy, Robot 3 chooses actions greedily with probability $0.851 = (1 - \frac{1}{6.7})$ and randomly with probability $0.149 = \frac{1}{6.7}$.

At first, the individual Q-table is empty. So, Robot 3 makes a random move from 2 of the valid moves (left and up). Here, the random move is chosen as left.

Individual Q-table for Robot 3 is updated as follows.

$$\begin{aligned} Q((2,-3),down) &= Q((2,-3),left) + \alpha \times (reward + \gamma \times \max_b(Q((1,-3),b))) \\ &= 0 + 0.525 \times (\frac{1}{9+1} + 0.7 \times 0) \\ &= 0.0525 \end{aligned}$$

Step 1 - Robot 4

According to epsilon greedy strategy, Robot 4 chooses actions greedily with probability $0.851 = (1 - \frac{1}{6.7})$ and randomly with probability $0.149 = \frac{1}{6.7}$.

At first, the individual Q-table is empty. So, Robot 4 makes a random move from 2 of the valid moves (right and up). Here, the random move is chosen as up.

Individual Q-table for robot 4 is updated as follows.

$$\begin{aligned} Q((-2,-3),up) &= Q((-2,-3),up) + \alpha \times (reward + \gamma \times \max_b(Q((-2,-2),b))) \\ &= 0 + 0.95 \times (\frac{1}{9+1} + 0.7 \times 0) \\ &= 0.095 \end{aligned}$$

Step 2 - Robot 1

According to epsilon greedy strategy, Robot 1 chooses actions greedily with probability $0.706 = (1 - \frac{1}{3.4})$ and randomly with probability $0.294 = \frac{1}{3.4}$.

In this case, Robot 1 chooses the σ action randomly. Now, as σ action represents a state where coordination is required, Robot 1 calculates its closest peer according to Manhattan distance. If, multiple peers satisfy the closeness criteria, the peer nearest to its goal is selected.

Table 4.4: Distances from Robot 1

Distance from Robot 1		
Robot 2	Robot 3	Robot 4
4	8	4

Table 4.5: Distances from Goal 1

Distance from Goal 1	
Robot 2	Robot 4
5	5

As all the minimum distances are equal, Robot 4 is randomly chosen. The peer Q-table is empty, and so, Robot 1 chooses action randomly as right. Now, peer Q-table is updated as below.

$$Q(((-2,2),(-2,-2)),right) = Q(((-2,2),(-2,-2)),right) + \alpha \times (reward + \gamma \times \max_b(Q((-1,2),b)))$$
$$= 0 + 0.84 \times (\frac{1}{8+1} + 0.7 \times 0)$$
$$= 0.09333$$

Then, individual Q-value is updated for σ action as follows.

$$Q((-2,2),\sigma) = Q((-2,2),\sigma) + \alpha \times (reward + \gamma \times \max_b(Q((-1,2),b)))$$
$$= 0 + 0.84 \times (\frac{1}{8+1} + 0.7 \times 0)$$
$$= 0.09333$$

Step 2 - Robot 2

According to epsilon greedy strategy, Robot 2 chooses actions greedily with probability $0.851 = (1 - \frac{1}{6.7})$ and randomly with probability $0.149 = \frac{1}{6.7}$.

In this case, Robot 2 chooses the σ action randomly. Now, as σ action represents a state where coordination is required, Robot 2 calculates its closest peer according to Manhattan distance. If, multiple peers satisfy the closeness criteria, the peer nearest to its goal is selected.

As all the minimum distance is 3, Robot 1 is chosen.

As the peer Q-table of Robot 2 is empty, Robot 2 chooses action randomly as

Table 4.6: Distances from Robot 2

Distance from Robot 2		
Robot 1	Robot 3	Robot 4
3	6	8

right. Now, peer Q-table is updated as below.

$$\begin{aligned} Q(((2,2),(-1,2)),down) &= Q(((2,2),(-1,2)),down) + \alpha \times (reward + \gamma \times \\ &\qquad \max_b(Q((2,1),b))) \\ &= 0 + 0.525 \times (\frac{1}{8+1} + 0.7 \times 0) \\ &= 0.05833 \end{aligned}$$

Then, individual Q-value is updated for σ action as follows.

$$\begin{aligned} Q((2,2),\sigma) &= Q((2,2),\sigma) + \alpha \times (reward + \gamma \times \max_b(Q((2,1),b))) \\ &= 0 + 0.525 \times (\frac{1}{8+1} + 0.7 \times 0) \\ &= 0.05833 \end{aligned}$$

Step 2 - Robot 3

According to epsilon greedy strategy, Robot 3 chooses actions greedily with probability $0.851 = (1 - \frac{1}{6.7})$ and randomly with probability $0.149 = \frac{1}{6.7}$.

Robot 3 chooses to select action greedily. However, as there is no associated Q-value with state (1,-3), it makes a random move of up.

Individual Q-table for Robot 3 is updated as follows.

$$\begin{aligned} Q((1,-3),up) &= Q((1,-3),up) + \alpha \times (reward + \gamma \times \max_b(Q((1,-2),b))) \\ &= 0 + 0.525 \times (\frac{1}{8+1} + 0.7 \times 0) \\ &= 0.05833 \end{aligned}$$

Step 2 - Robot 4

According to epsilon greedy strategy, Robot 4 chooses actions greedily with probability $0.851 = (1 - \frac{1}{6.7})$ and randomly with probability $0.149 = \frac{1}{6.7}$.

Robot 4 makes a random move from 4 of the valid moves (up, right, left and down). Here, the random move is chosen as left.

Individual Q-table for Robot 4 is updated as follows.

$$\begin{aligned} Q((-2,-2), left) &= Q((-2,-2), left) + \alpha \times (reward + \gamma \times \max_b(Q((-3,-2), b))) \\ &= 0 + 0.95 \times (\frac{1}{10+1} + 0.7 \times 0) \\ &= 0.08636 \end{aligned}$$

Continuing this update procedure, 1500 episodes (a single episode ends when all the robots reach their goals) are run for the training. Another scenario in training is when a collision occurs. To see this, the next two steps of the algorithm for Robot 1 and 2 are given below.

Step 3 - Robot 1

According to the epsilon-greedy strategy, Robot 1 chooses actions greedily with probability $0.706 = (1 - \frac{1}{3.4})$ and randomly with probability $0.294 = \frac{1}{3.4}$.

Robot 1 chooses to act greedily. However, as the Q-table for state (-1, 1) is empty, it acts randomly and chooses right.

Individual Q-table for Robot 1 is updated as follows.

$$\begin{aligned} Q((-1,1), right) &= Q((-1,1), right) + \alpha \times (reward + \gamma \times \max_b(Q((0,1), b))) \\ &= 0 + 0.84 \times (\frac{1}{6+1} + 0.7 \times 0) \\ &= 0.12 \end{aligned}$$

Step 3 - Robot 2

According to epsilon greedy strategy, Robot 2 chooses actions greedily with prob-

ability $0.851 = (1 - \frac{1}{6.7})$ and randomly with probability $0.149 = \frac{1}{6.7}$.

In this case, Robot 2 chooses the σ action randomly. Now, as σ action represents a state where coordination is required, Robot 2 calculates its closest peer according to Manhattan distance. If, multiple peers satisfy the closeness criteria, the peer nearest to its goal is selected.

Clearly, the closest peer towards goal is Robot 1. As the peer Q-table of Robot 2 for ((1,1),(0,1)) is empty, robot 2 chooses action randomly as left. Now, a conflict occurs and Robot 2's position does not change. It receives a constant penalty of 20. The peer Q-table is updated as below.

$$\begin{aligned} Q(((1,1),(0,1)),left) &= Q(((1,1),(0,1)),left) + \alpha \times (reward + \gamma \times \\ &\qquad \max_b(Q((1,1),b))) \\ &= 0 + 0.525 \times (-20 + 0.7 \times 0) \\ &= -10.5 \end{aligned}$$

Then, individual Q-value is updated for σ action as follows.

$$\begin{aligned} Q((1,1),\sigma) &= Q((1,1),\sigma) + \alpha \times (reward + \gamma \times \max_b(Q((1,1),b))) \\ &= 0 + 0.525 \times (100 + 0.7 \times 0) \\ &= 52.5 \end{aligned}$$

4.5 Summary

An IDMG-based decentralized self-adaptive system has been proposed that uses a single peer agent to address partial knowledge and distinguishes coordinating and non-coordinating states to reduce coordination overhead. An improved version of traditional multiagent Q-learning, IDMG-based Q-learning is proposed. It adds an additional σ action to learn the coordinating states using negative rewards.

Multiagent Q-learning is performed with a single peer in these states where individual Q-learning is used in the other states. Prior to this, reward functions are defined according to goal types. For multi-objective self-adaptive systems, a weighted sum of the rewards is used. These weights are adjusted at runtime to prioritize the violated goals.

Chapter 5

Implementation and Result Analysis

This chapter contains the implementation details, experiments and result analysis of the proposed approach. The discussion of the results to explain some important insights are also presented. The technique was implemented in Java and validated using the adaptive robot navigation problem and the Tele Assistance System (TAS). The average rewards were compared among the proposed technique, random action selection, individual Q-learning and joint Q-learning. The convergence speed was also compared. It was seen that the proposed technique converged faster as it provided higher rewards in early runs. Different agent selection algorithms such as random, round robin and heuristic-based were compared. It was seen that heuristic-based one performed better. This demonstrates the requirement of a problem-specific agent selection technique. Overall, The incorporation of partial knowledge was effective with fast convergence and coordination overhead was reduced due to coordinating in only selected states.

5.1 Implementation

In this section, the tools and technologies of implementing the proposed technique are discussed. The class diagram of the implementation is also provided to get an overall view of the technique.

5.1.1 Tools and Technologies

The proposed technique was implemented and incorporated into the two aforementioned systems. During the implementation, some tools and technologies were used which are described below.

- Java Development Kit 8 (JDK 8)[1]: Java Development Kit is the development environment for Java-based applications. JDK 8 is written to support Java 8 which has some advanced functionalities such as Stream API, lambda expressions, improved Collections API etc. Due to these advanced functionality, JDK 8 is used for the implementation.

- Eclipse Oxygen[2]: Eclipse is an open-source Integrated Development Environment (IDE) for Java. Eclipse Oxygen is the recently released version of Eclipse that supports faster debugging, source code differencing, better source code analysis support etc. It was used to write, debug and execute the source code of the proposed technique.

- Apache Commons IO[3]: It provides a simpler interface to perform IO related task in Java. The Q-values are frequently written and read from files. This library has been used for this purpose.

- SAXParser[4]: This API is used to read, write and parse through XML docu-

[1]http://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html
[2]https://eclipse.org/oxygen/
[3]https://commons.apache.org/proper/commons-io/
[4]https://docs.oracle.com/javase/7/docs/api/org/xml/sax/package-summary.html

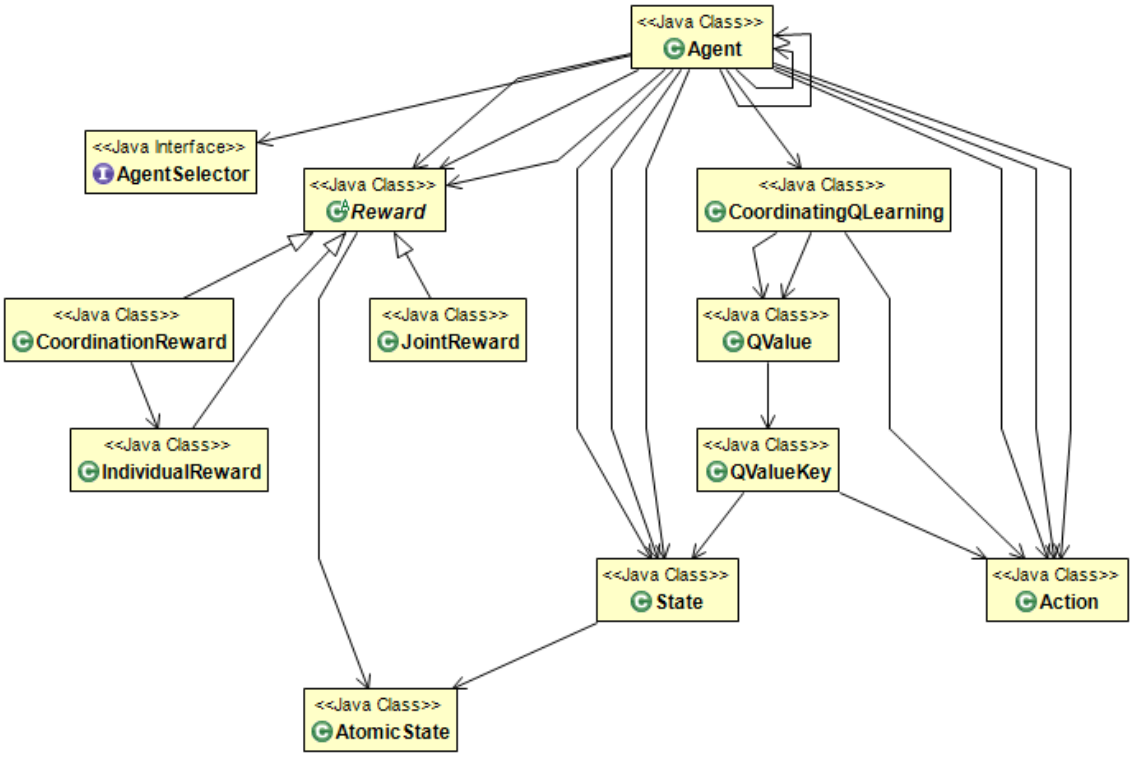


Figure 5.1: Class Diagram of the Proposed Technique

ments in Java. The parameters in the proposed technique such as λ, coordination reward etc. are read from an XML document so that these could be tuned externally. The SAXParser provides an easy to use API for this purpose. It is also memory efficient which is required as the in-memory Q-tables in the implementation consume a lot of memory.

5.1.2 Class Diagram

Figure 5.1 shows the class diagram of the proposed technique. The *Agent* class is the control loop which implements Algorithm 4.1. *CoordinatingQLearning* class implements the individual Q-learning and peer Q-learning (Q-learning with peer agents from Algorithm 4.1). *CoordinatingQLearning* contains two lists of *QValue* class that represents the individual and peer Q-tables. *QValue* class contains an instance of *QValueKey* which consists of *State* and *Action* class instances.

The *Reward* class calculates three types of rewards which are implemented in

CoordinationReward, *JointReward* and *IndividualReward* classes. The first one is for marking coordinating states, the second one calculates global goal conformance and the third one calculates local goal conformance. The *Reward* class aggregates all the rewards by calculating a weighted sum. The *Reward* class is also responsible for dynamic weight update. The *AgentSelector* interface is used to implement the problem-specific agent selection algorithm.

5.2 Adaptive Robot Navigation

This is a famous problem used widely in the literature [5, 7, 8, 9, 81]. Although this problem is largely used in the multiagent system domain, it directly fits into the decentralized self-adaptive system mechanism. As shown in Figure 5.2, four robots are on four corners of the grid. These need to reach their goals on the opposite sides. However, two cannot pass the doorway at the same time. Here, each robot can be considered as a managed system. A control loop can be attached to each of these which manages their routes. The target of these control loops is to reach the goal without collision following the shortest route.

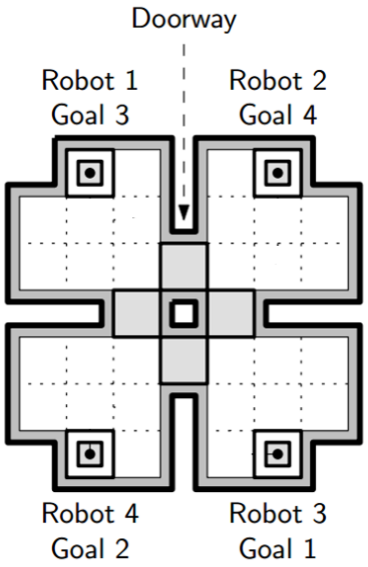


Figure 5.2: Adaptive Robot Navigation inside a Grid

5.2.1 Problem Definition

To apply the proposed technique to this problem, at first, state, action and reward functions are defined.

- State: State consists of the position of a robot in the Cartesian coordinate, as shown in the previous chapter.

- Action: A robot can move right, left, top or bottom. A valid action set in a specific state must not contain any action that leads to a collision with the wall. For example, right is not a valid action for Robot 2 in state (2,3) (Figure 5.2).

- Reward function: The local goal of each robot is to reach the goal following the shortest path. The global goal is to avoid any conflict with another robot. The local goal is a minimization goal as the distance between a robot's position and the goal must be minimized. So, it is defined as below in Equation 5.1.

$$R^i_< = \frac{1}{d_i + 1} \tag{5.1}$$

Here, d_i is the distance between the robot and its goal. The distance is calculated using Manhattan distance [80]. Manhattan distance is the summation of horizontal and vertical distance between two points. For example, the Manhattan distance between 1,1 and 2,2 is 2.

The global goal can be stated as a threshold-based goal which is - the distance between two robots must be greater than 0 in doorways. This can be restated as below to derive the threshold-based global reward function.

$$\begin{aligned} & d_{i,j} > 0 \\ \iff & -d_{i,j} < 0 \\ \iff & -d_{i,j} < 1 \end{aligned} \tag{5.2}$$

Here, $d_{i,j}$ is the distance between robot i and j. Using Equation 5.2 and 4.4,

R_θ is defined as follows.

$$R_\theta = \begin{cases} \dfrac{2 + d_{i,j}}{2} & \text{if } 1 + d_{i,j} > 0 \\ \dfrac{1}{2} - \dfrac{2 + d_{i,j}}{2 \times (2 + d_{i,j}^{max})} & \text{if } 1 + d_{i,j} < 0 \\ .\dfrac{1}{2} & \text{if } \theta - m = 0 \end{cases} \tag{5.3}$$

Equation 5.3 is used to calculate the global goal values directly using Equation 4.10.

After the state, action and reward functions are specified, the proposed technique is executed and experiments are conducted to test its effectiveness.

5.2.2 Experimental Analysis

The technique can be called an effective one if both the local and global goals are continuously satisfied. Moreover, it can be called efficient if it learns the optimal solution faster than the joint Q-learning solution. In the following sub-sections, the experiments to test and compare these are described in details. The result analysis and discussion are presented to examine whether this technique reduces coordination overhead and learns optimally with partial knowledge.

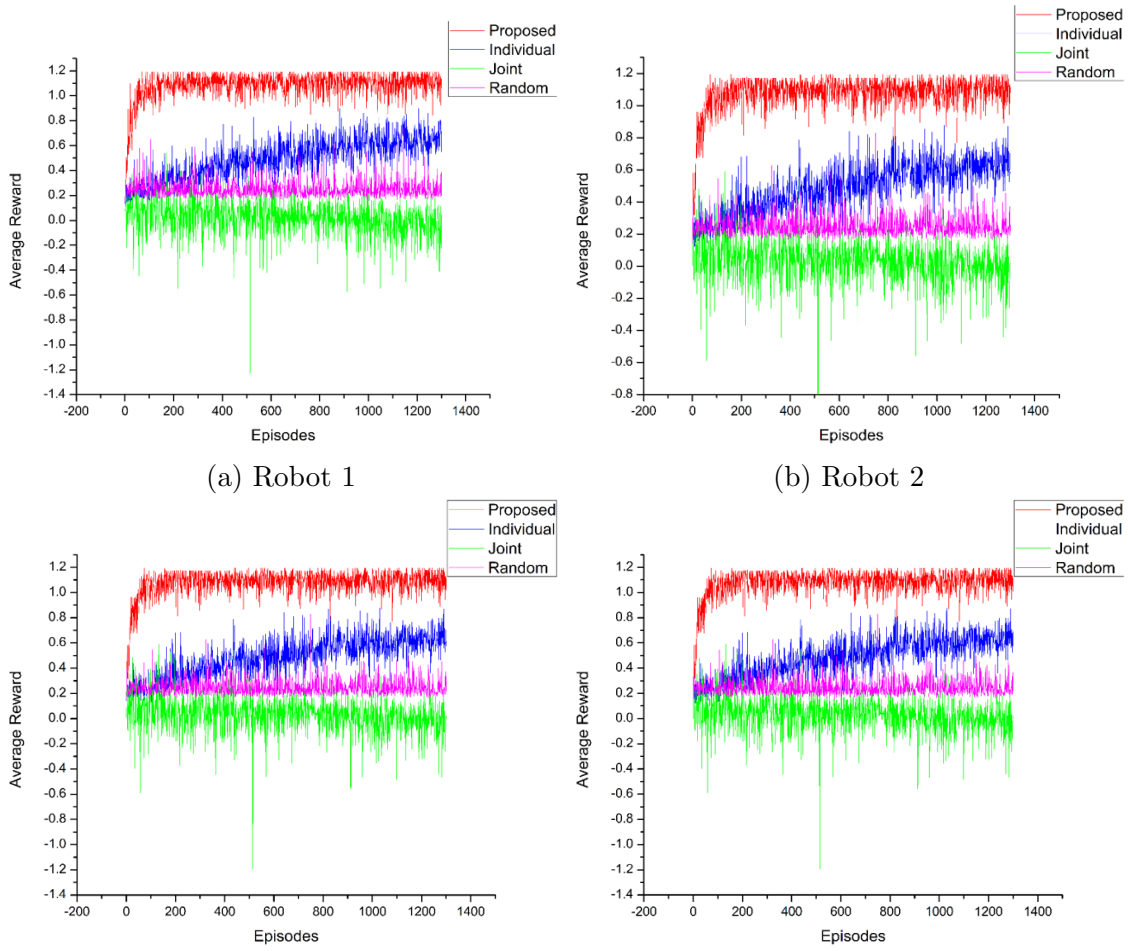
Experimental Setup

At first, the optimal parameter values were determined. The learning rate (α) was decayed starting from decay factor 1.05, increasing by 0.00085 at every episode. The epsilon value (ϵ) was also decayed using a decay factor starting from 1, increasing by 0.005 at every episode up to 10. The discount factor (λ) was chosen as 0.9. The coordination reward (r_c) value was chosen as 100. A constant reward of 10 was also provided when goal state was reached. All the parameter values were chosen empirically by experimenting with the system.

The adaptation effectiveness of the technique is compared with three other approaches which are individual Q-learning, joint Q-learning and random action selection. Following the literature, adaptation effectiveness was observed directly by comparing the rewards [5, 7, 19, 81]. The number of collisions was also compared to observe whether the global goal is satisfied. The efficiency of the approach was validated comparing the average timesteps inside an episode. It indicates how fast the approach learns the optimal actions. The coordination overhead of the approach has been discussed at this point. The communication overhead due to agent selection was compared with joint Q-learning. Finally, different agent selection techniques (static, random, round robin, nearest neighbour nearest goal) were compared to observe whether agent selection techniques have any significant impact on convergence.

Result Analysis

Figure 5.3 shows the comparison of rewards among the four approaches. These figures are almost similar as the agents are homogeneous. It is visible that the proposed approach has much higher reward than the other approaches. The reward is initially low and similar to the other approaches. Near 50 episodes, the rewards from the proposed approach start to increase. Approximately at 200 episodes, rewards become almost constant which indicates convergence. The rewards from individual Q-learning keeps increasing at this point. Individual Q-learning has lower number of state spaces to learn and so, should have converged faster. Nevertheless, the individual Q-learning converges locally but global convergence is not achieved due to lack of coordination. The reward is also much less than that of the proposed method. The joint Q-learner progresses very slowly. In fact, the reward values seem to decrease which is counterintuitive. It happens due to two reasons. Firstly, the joint Q-learning has much larger state space to learn and so, the Q-table is highly incomplete during these episodes. Secondly, most of the



(a) Robot 1

(b) Robot 2

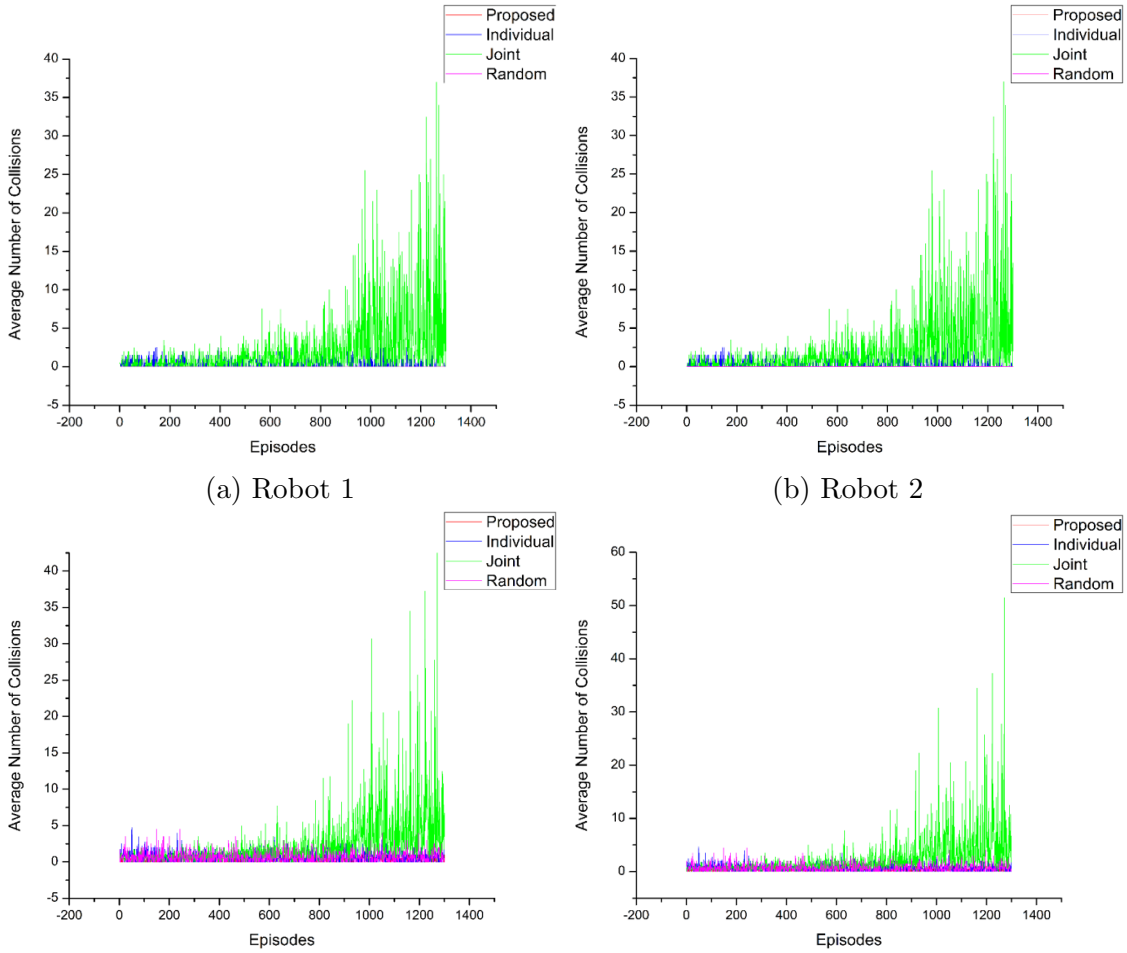
(c) Robot 3

(d) Robot 4

Figure 5.3: Comparison of Average Reward Per Episode

actions are randomly taken and the learning rate is very high for joint Q-learning during these episodes. This is due to the larger state space of the joint-Q learning algorithm, learning rate and ϵ decays slowly. The joint Q-learning should converge to an optimal solution much later than the proposed technique which prunes the state space separating coordinating states and considering single peer agent. The random action selection, which is chosen as the baseline, performs poorly.

The comparison of average number of collisions per episode among these four techniques is given in Figure 5.4. The figure shows almost similar result to Figure 5.3. The number of collisions in joint Q-learning is much higher due to aforementioned reasons. Individual Q-learning and random action selection also produce solutions with collisions. The proposed technique initially yields a solution with



(a) Robot 1 (b) Robot 2

(c) Robot 3 (d) Robot 4

Figure 5.4: Comparison of Average Number of Collisions Per Episode

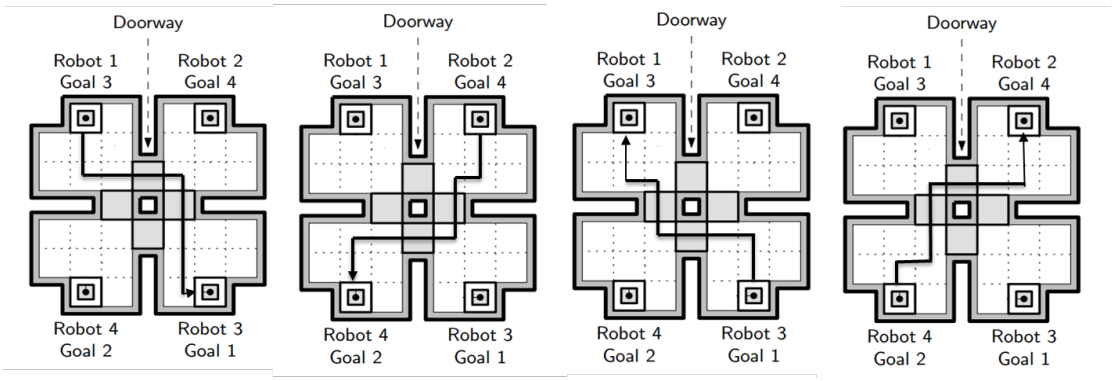


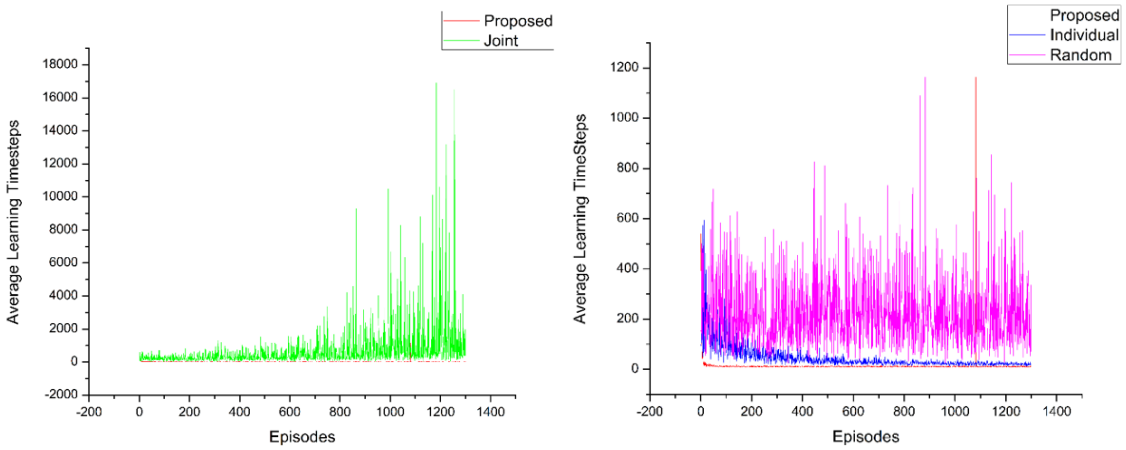
Figure 5.5: Paths Taken by Each Robot after Learning is Conducted for 1500 Episodes

collisions but produces collision-free solutions soon as it converges.

Table 5.1 shows the average performance of all these four techniques after 1500 episodes of training. The proposed technique produces the highest reward and no

Table 5.1: Average Performance after 1500 Episodes of Training

Technique	**Average Reward**	**Miscoordination/ collisions**
Individual	0.1952822	2
Joint	0.1609553	4
Proposed	0.21433	0
Random	0.150962	6



(a) Robot 1 - Proposed vs Joint Q-learning (b) Robot 1 - Proposed vs Individual vs Random

Figure 5.6: Comparison of Average Timesteps for Learning Per Episode

collisions occur. So, it learns faster than the other methods to reach the optimal point. To observe whether convergence is achieved, Figure 5.5 shows the routes of each robot after 1500 episodes of training. It is seen that each robot needs 10 steps to reach the goal. As the minimum distances between the initial positions of each robot and their goal states are 10, it can be concluded that the proposed technique has converged to the optimal solution. Both the local goal and global goal are achieved and so, decentralized self-adaptation with partial knowledge is shown to be effective.

The average timesteps for learning per episode are compared in Figure 5.6. Here, timestep indicates a single execution step in every episode. Larger timesteps to goal measure distance from the most optimal solution. It is seen from Figure 5.6(a) that joint Q-learning is far away from convergence and produces nonoptimal solution in every episode. The proposed technique performs much better

Table 5.2: Agent to Agent Communication Overhead of the Proposed Method and Joint Q-learning

Robots	**Avg. Communication Delay (Milliseconds)**	
	Proposed	**Joint**
First	253.9993	374.3384
Second	242.1929	374.5943
Third	239.8981	374.6212
Fourth	252.7602	374.4747

Table 5.3: Total Coordination Overhead Per Episode of the Proposed Method and Joint Q-learning

Robots	**Total Coordination Delay (Milliseconds)**	
	Proposed	**Joint**
First	39566	73751
Second	27294	106122
Third	38427	101145
Fourth	45725	95269

reaching the goal within 10-20 timesteps per episode. Figure 5.6(b) compares it with the other two approaches. These perform worse than the proposed technique. Individual Q-learning performs worse because it leads to collision for avoiding coordination. As a robot cannot move into the doorway when a collision occurs, larger timesteps are taken to reach the goal.

Table 5.2 shows the agent to agent communication delay of the proposed approach and the joint Q-learning technique. It is observed that the proposed technique has lower overhead. This is because each agent needs to communicate their state for agent selection in the proposed approach in coordinating states. In the joint Q-learning, agents communicate their states and actions in all the states. It can be stated from this result that the proposed approach reduces coordination overhead at the cost of the average agent to agent communication overhead increase. The coordination overhead reduction is also visible from table 5.3. The total coordination delay of the proposed approach within an episode is much less than the joint Q-learning approach. This is because coordination is not required in all the states in the proposed technique.

Figure 5.7 shows the comparison of reward for three agent selection algorithms

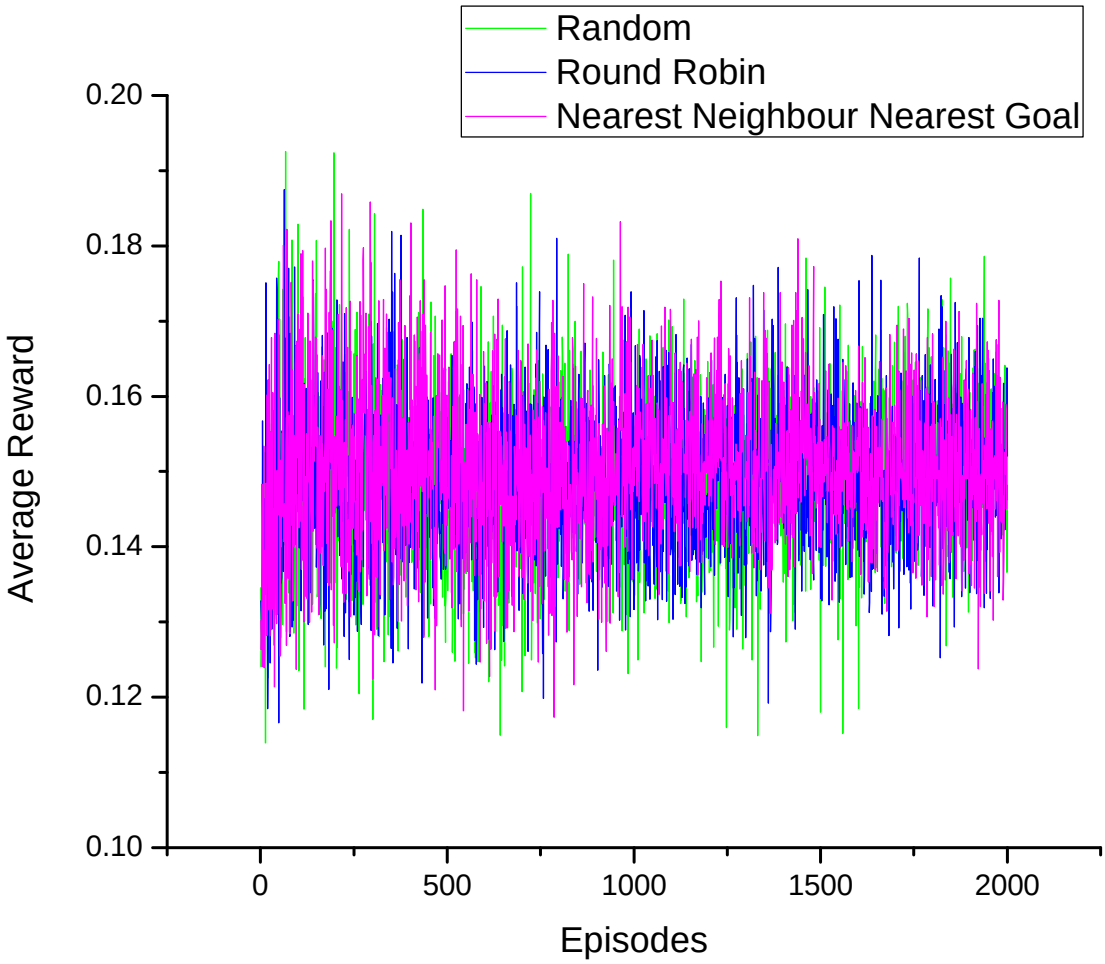
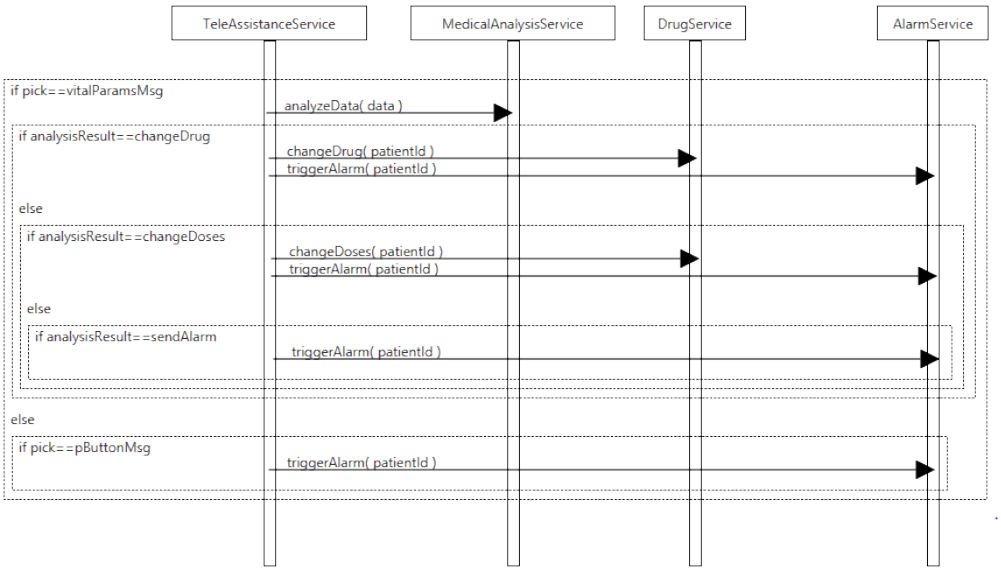


Figure 5.7: Comparison of Reward for Different Agent Selection Algorithms

which are random, round robin and nearest neighbour nearest goal. The first one chooses a peer agent randomly. Round robin selects a peer agent one by one in turn. Nearest neighbour nearest goal is a heuristic-based approach. In this technique, the nearest neighbour is considered as the peer agent. If there are two nearest neighbours, the one that is the closest to the agent's goal is taken. The figure depicts that this agent selection technique performs better than the other three. A Wilcoxon rank sum test was conducted to see whether the difference is significant. Here, the null hypotheses are that the rewards from the nearest neighbour nearest goal is less than those of random and round robin agent selection methods. For these two hypotheses, the p-values are 2.12e−5 and 3.572e−6. So, the null hypotheses can be rejected. It means that the heuristic-based approach



(a) The Workflow Diagram of the Tele Assistance System (TAS)

```
START [patientId,pick]

    // Declaration of pick actions
    vitalParamsMsg = 1
    pButtonMsg = 2
    stopMsg = 3

    // Declaration of analysisResults
    changeDrug = 1
    changeDoses = 2
    sendAlarm = 3

    if (pick == vitalParamsMsg){
        data = this.getVitalParameters()
        analysisResult = MedicalAnalysisService.analyzeData(data)

        if (analysisResult == changeDrug){
            DrugService.changeDrug(patientId)
            AlarmService.triggerAlarm(patientId)
        }
        else if (analysisResult == changeDoses){
            DrugService.changeDoses(patientId)
            AlarmService.triggerAlarm(patientId)
        }
        else if (analysisResult == sendAlarm)
            AlarmService.triggerAlarm(patientId)
    }
    else if (pick == pButtonMsg) {
        AlarmService.triggerAlarm(patientId)
    }

RETURN
```

(b) The Pseudocode of the TAS Workflow

Figure 5.8: The workflow of the Tele Assistance System (TAS)

outperforms the generic approaches. This is why the agent selection algorithm was kept problem specific.

5.3 Tele Assistance System (TAS)

Tele Assistance System (TAS) was first proposed by Weyns et al. as a model problem for self-adaptive systems [4]. It was developed under the "A Foundation for Engineering Decentralized Self-Adaptive Software Systems - FEDerAteS" project [82]. It was accepted as a standard model problem of self-adaptation in Software Engineering for Self-Adaptive Systems (SEAMS) repository [83]. The existing TAS implementation is centralized. So, it was modified to support decentralized adaptation by adding a control loop to each service. The implementation was done in JAVA and it utilizes the Research Service Platform (ReSeP) for service-based systems by Weyns et al. [84]. Each local control loop contains reward calculation, state data reader, action selection and IDMG-Based Q-learning components. The reward calculation component uses current state information (i.e., current values of the metrics) to calculate composite reward values using Equation 4.1. The state data reader component collects current values of the metrics and encodes these to get the current state. When σ is selected, this component requests and fetches current state data from a predefined peer. The IDMG-Based Q-learning and the action selection component implements Algorithm 4.1 and 4.2 respectively.

Figure 5.8 shows the workflow of the TAS services. The *TeleAssistanceService* controls the workflow centrally and manages adaptation. In the decentralized version, this service is eliminated and a control loop is added to each of the other services. From Figure 5.8(b), *MedicalAnalysisService* is dependent on both *AlarmService* and *DrugService*. The peers were selected based on this dependency. The peers of *MedicalAnalysisService*, *AlarmService* and *DrugService* were *DrugService*, *MedicalAnalysisService* and *AlarmService* respectively.

The goal specification of TAS is given below. These goals are a more definite version of the ones presented in Table 4.1.

Global Goals

- At least one service must have failure rate less than 0.12
- Average cost per service must be minimized

Medical Analysis Service Local Goals

- Each service failure rate must be less than 0.18
- Each service must have response time less than 5 seconds

Alarm Service Local Goals

- Each service failure rate must be less than 0.14
- Each service must have response time less than 6 seconds

Drug Service Local Goals

- Each service failure rate must be less than 0.15
- Each service must have response time less than 12 seconds

5.3.1 Problem Definition

The state, action and reward functions for TAS are defined as follows.

- State: TAS operates in continuous state space which must be discretized. The state space encoding for discretization is shown in Table 5.4. The state space is encoded based on metric values and goals. For example, Drug Service has local and global failure rate thresholds of 0.15 and 0.12 respectively. So, the state space is encoded by defining $failure\ rate \leq 0.12$ as low, $0.12 < failure\ rate \leq 0.15$ as medium and $0.15 < failure\ rate \leq 1$ as high. The service cost goal is a minimization goal and so, does not have any goal

Table 5.4: State Space Encoding for TAS

Goal Attribute	Service Name	Value Range	Corresponding Category
Service Cost	Medical Analysis Service	[0, 2.15]	Low
		(2.15, 4]	Medium
		(4, ∞)	High
	Alarm Service	[0, 2]	Low
		(2, 4]	Medium
		(4, ∞)	High
	Drug Service	[0, 10]	Low
		(10, 13]	Medium
		(13, ∞)	High
Failure Rate	Medical Analysis Service	[0, 0.12]	Low
		(0.12, 0.18]	Medium
		(0.18, 1]	High
	Alarm Service	[0, 0.12]	Low
		(0.12, 0.14]	Medium
		(0.14, 1]	High
	Drug Service	[0, 0.12]	Low
		(0.12, 0.15]	Medium
		(0.15, 1]	High
Individual Service Response Time	Medical Analysis Service	[0, 5]	Low
		(5, ∞)	High
	Alarm Service	[0, 6]	Low
		(6, ∞)	High
	Drug Service	[0, 5]	Low
		(5, ∞)	High

threshold. It is encoded based on the observed range of values and tuned manually. For example, it was seen that the technique performs best when Drug Service cost is encoded as given in Table 5.4.

- Action: Medical Analysis Service has three actions which are *MedicalAnalysisService1*, *MedicalAnalysisService2* and *MedicalAnalysisService3*. Alarm Service and Drug Service similarly have 3 and 2 actions respectively. These actions are of different configurations, that is, these have different service cost, failure rate and response time.

- Reward Function: The reward function design of TAS is straightforward. The local and the first global threshold-based goals are defined according to Equation 4.4 and 4.10 respectively. The cost minimization goal is defined using Equation 4.5.

5.3.2 Experimental Analysis

In this section, the experimental setup, the result and the discussion are provided. TAS is used to test the effectiveness of the proposed approach in multi-objective self-adaptive systems. The experiments were conducted with this objective in mind.

Experimental Setup

The parameter value selection of TAS is similar to that of adaptive robot navigation. TAS is a multi-objective system. As mentioned earlier, the main goal of experimenting on TAS was to observe how the technique performs in multi-objective self-adaptive systems. Three types of experiments were conducted. Firstly, the total average reward achieved using the proposed technique was compared with three other techniques which are random, individual Q-learning and joint Q-learning. Secondly, the learning time was compared between the proposed approach and the joint Q-learning technique. Finally, the proposed technique was compared with and without weight update.

Result Analysis

Figure 5.9 compares the total observed reward from the proposed technique, random action selection and individual Q-learning. It is seen that the reward values from the proposed approach are mostly higher and more stable than the other two approaches. The reward values from random action selection vary a lot which is obvious due to the randomness in this technique. The variation in reward values is also higher in the individual Q-learning approach than the proposed one. Moreover, up to about 600 iterations, these three approaches perform almost similarly. However, the proposed approach results in overall higher total reward values in the subsequent iterations. A Wilcoxon rank sum test was performed to see whether the mean total reward value in the proposed approach is significantly higher than

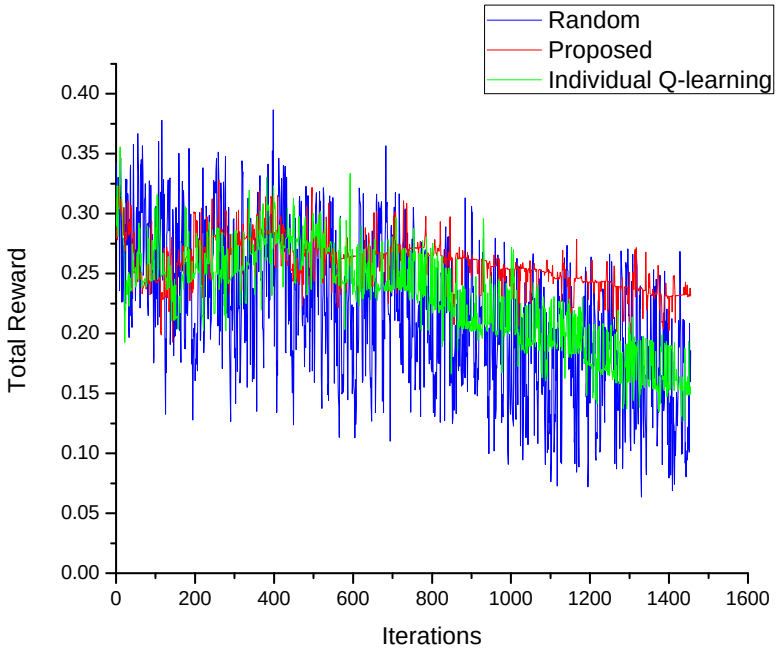


Figure 5.9: Comparison of Total Rewards Obtained among the Proposed Approach, Random Action Selection and Individual Q-learning

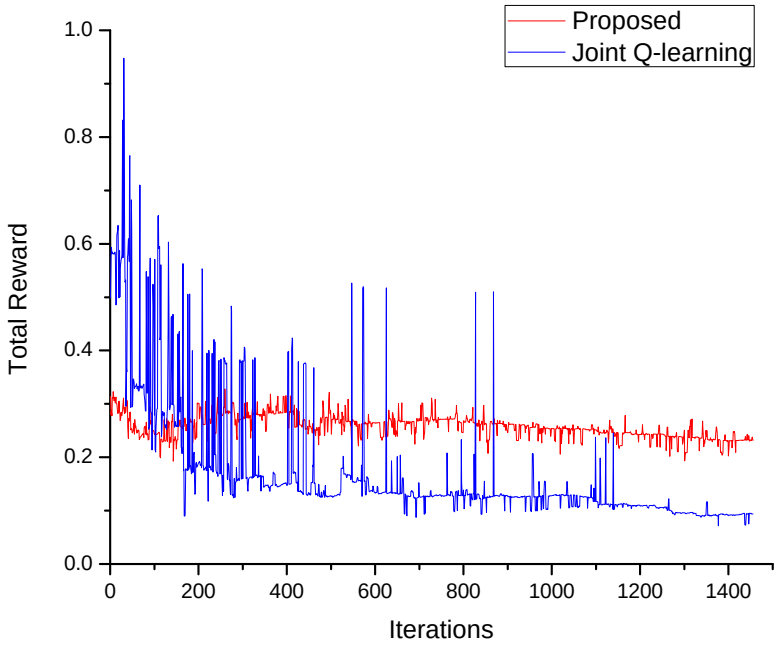


Figure 5.10: Comparison of Rewards from the Proposed Method and Joint Q-learning

the other two approaches [85]. The p-values (1.14e−119 for random and 7.95e−117 for individual Q-learning) less than the significance level 0.05 in both cases indicate that the reward values obtained from the proposed approach are significantly

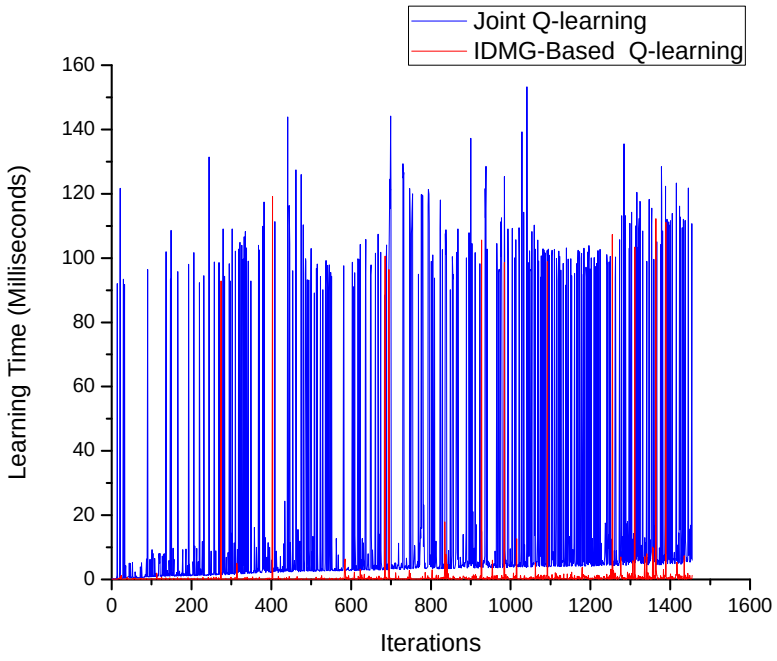
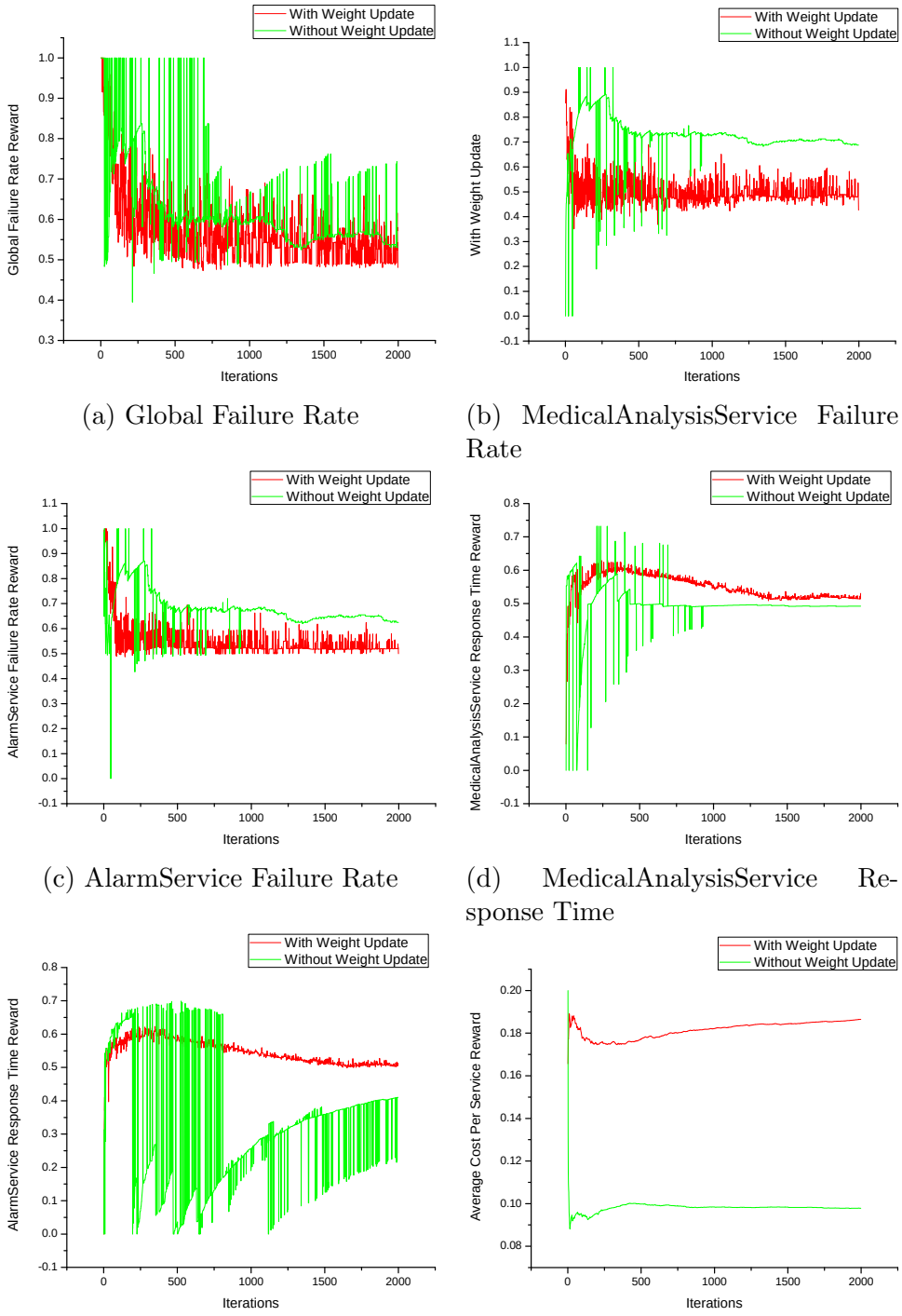


Figure 5.11: Comparison of Learning Time from the Proposed Method and Joint Q-learning

higher. This is because coordination is taken into consideration in coordinating states where individual Q-learning and random action selection do not consider coordination at all.

Figure 5.10 shows the comparison of total rewards from the proposed approach using IDMG-Based Q-learning and joint Q-learning. The joint Q-learning approach considers global knowledge (i.e., all the locally managed systems) for learning. So, it should result in higher total reward than the proposed partial knowledge-based approach. The figure shows that, the total reward from the joint Q-learning is higher up to about 200 iterations, but it is counterintuitively lower afterwards in most cases. This can be explained as follows. Each local control loop needs to maintain a large table of Q-values for large global state space. So, the joint Q-table is mostly incomplete in earlier runs as discussed previously. The initial actions selected by the joint Q-learning approach is mostly nonoptimal. This further delays convergence as the rewards decrease slowly in subsequent iterations (Figure 5.10). This situation resembles the timeliness-impact of making



(a) Global Failure Rate

(b) MedicalAnalysisService Failure Rate

(c) AlarmService Failure Rate

(d) MedicalAnalysisService Response Time

(e) AlarmService Response Time

(f) Average Cost Per Service

Figure 5.12: Comparison between the proposed approach with and without weight update

adaptation decisions described by Weyns et al. [4]. This situation is confirmed by Figure 5.11 which shows the learning time of the two approaches. The learning time of the proposed approach is much lower in most cases. The few cases where

the proposed approach has higher learning time were analyzed. It was observed that high learning time occurred either because the corresponding state was a coordinating state or due to implementation platform issues (e.g., JVM garbage collection).

Figure 5.12 shows the values of the reward functions considering whether weight update is applied. In Figure 5.12(a), 5.12(b) and 5.12(c), the proposed approach produces much higher values when weights are not updated. However, the technique without weight update performs much worse in Figure 5.12(d), 5.12(e) and 5.12(f). Although in Figure 5.12(d), the reward values without weight update are slightly lower than the ones with weight update, in the other two figures, these are much lower. Moreover, Figure 5.12(d) and 5.12(e) indicate that the reward values without considering weight update always remain under 0.5. It means that this approach cannot adapt the response time goals though it performs much better for other goals. The weight update scheme helps to ensure that every goal is satisfied by providing more weight to the violated goal which requires adaptation. This is why, from the figures, all goals are continuously satisfied by the proposed approach considering weight update. This shows the effectiveness of the weight update mechanism.

5.4 Discussion of the Results

The proposed approach was tested on an adaptive robot navigation problem and a tele assistance system. In the first one, the approach was observed to be more effective than random action selection, individual Q-learning and joint Q-learning. Although joint Q-learning considers all the agents and so, should find the optimal action selection, it takes a long time to do so. This is because the number of possible entries in the joint Q-table is large and all these need to be visited infinitely often for convergence [58]. The proposed approach reaches the optimal point much

faster as the number of Q-table entries is much smaller. Similar higher rewards for the proposed approach are seen for TAS.

The high effectiveness of the approach is achieved due to three reasons which are agent homogeneity assumption, heuristic-based agent selection and coordination state learning. Firstly, in the previous chapter, it was assumed that agents are homogeneous. If this is the case, any agent can be dynamically used as peer agent irrespective of their identity, that is, agent selection can be performed. Otherwise, when peer agents are dynamically changed, the learned knowledge using a specific peer agent cannot be utilized with another peer agent. Secondly, the agent selection should be problem-specific. Figure 5.7 shows the validity of this statement. From Figure 5.7 and the performed Wilcoxon rank sum test, random selection performs worse than round robin selection. This is easily explainable if the agent selection process is compared to a tournament. A tournament where randomly chosen team compete is less fair than the one where teams face one another in turns. The heuristic-based agent selection technique performs better than these two because it intuitively chooses the peer agent which has the highest probability of conflict. Generally, an agent is more probable to collide with its nearest agent and the agent within the path towards its goal. Finally, agents learn peer Q-values only in coordinating states. As a result, the state space is greatly pruned.

Another important thing to discuss is the agent selection overhead. Table 5.2 indicates that the proposed method has high communication overhead. In the worst case, the communication overhead can be similar to joint Q-learning. Nevertheless, the memory overhead for coordination is still reduced (smaller Q-table). Although the technique may have communication overhead similar to joint Q-learning, it converges faster.

Figure 5.12 from TAS shows that weight update is the main reason that the technique succeeds in multi-objective self-adaptive systems. More importantly,

the weight update is lightweight. Self-adaptation may otherwise take longer time and result in the timeliness impact described by Weyns et al. [4]. In the worst case, it may lead to thrashing. So, the proposed dynamic weight update approach is a better alternative to some heavyweight approaches (such as W-learning [33]).

5.5 Summary

This chapter presents the results of experiments conducted on two systems which are adaptive robot navigation and Tele Assistance System (TAS). The average rewards per episode have been compared with random action selection, individual and joint Q-learning. The proposed technique has produced higher rewards for both TAS and adaptive robot navigation problem. The number of collisions in the doorway has also been compared for the adaptive robot navigation. The proposed technique has been observed to converge into zero collisions. The learning timesteps of the proposed technique are also the lowest as the Q-table is smaller and coordination has been considered. Although agent selection communication overhead is higher, it is still lower than the joint Q-learning approach. In TAS, the average rewards using dynamic weight update has been compared with that of static weight assignment (i.e., without weight update). The result has shown that dynamic weight update helps to balance multiple goal conformance. As the proposed technique helps to dynamically self-adapt in a decentralized environment using only single peer agents and selecting specific states for coordination, convergence was faster than the traditional joint Q-learning approach.

Chapter 6

Conclusion

This thesis proposes a decentralized self-adaptation technique that incorporates partial knowledge and reduces coordination overhead. For this, an Interaction Driven Markov Game (IDMG) based approach has been devised that considers coordination only when lack of it leads to global goal violation. This coordination is performed with single agent to satisfy the assumption of partial knowledge. This chapter discusses the summary of the thesis with respect to its contribution and achievements. The threats to validity of this work are presented next. Finally, it is concluded with some future research directions.

6.1 Thesis Summary

The decentralization of self-adaptive systems is important due to the current outburst of distributed systems. Decentralized self-adaptive systems contain a control loop for each local subsystem in order to manage the adaptation process. These control loops have the objective to dynamically satisfy two types of goals which are local and global goal. Local goal is specific to the subsystem and global goal is the overall system objective. Coordination among multiple control loops is required to satisfy these goals. As a result, the control loops need to continuously monitor the adaptation decisions of one another to work correctly. This large sharing

of information harms scalability. To build a scalable decentralized self-adaptive system, partial knowledge, that is, the limited sharing and storage of such information should be the norm [4]. Incorporating partial knowledge also helps to reduce coordination overhead. In fact, a scalable and efficient decentralized self-adaptive system operates with as less coordination overhead as possible. Apart from all these, an effective decentralized self-adaptation technique should support multi-objective systems as most of the real-life systems are multi-objective ones.

As this thesis solves these problems, it makes three important contributions which are partial knowledge incorporation, coordination overhead reduction and multi-objective self-adaptation. The proposed technique is based on Interaction Driven Markov Game (IDMG), a game-theoretical framework from Multiagent System domain that separates coordinating and non-coordinating states [5]. This is different from the learning-based approaches in the literature such as [19, 21, 33] etc. which use Markov game. The whole decentralized self-adaptation problem is defined to fit into this framework and a learning algorithm incorporating partial knowledge is introduced. In the following subsections, these three contributions and achievements of the proposed technique are summarized.

6.1.1 Partial Knowledge Incorporation

At first, state, action and reward functions are defined. Reward functions are classified according to goals which are scope-based and target-based. Scope-based goals are local and global goals and target-based ones are divided into threshold-based, minimization and maximization goals. Reward functions are defined in a way that threshold-based goals retain goal violation detection property, that is, these function values also follow a threshold. More specifically, these reward functions provide a less than or equal to 0.5 value when a goal violation occurs. After providing the definitions, an algorithm is proposed to fit the problem into IDMG. This IDMG-based algorithm uses only a single control loop for coordi-

nation rather than all the control loops. Thus, it satisfies the partial knowledge criteria mentioned previously.

The proposed technique was validated using two well-known self-adaptive systems namely adaptive robot navigation and the Tele Assistance System (TAS). The states, actions and reward functions of these systems were defined and the parameter values were decided before the experiment. Then, the rewards from the proposed methodology were compared against three other methods which are random action selection, individual Q-learning and joint Q-learning. The proposed technique resulted in higher rewards and derived the optimal action selections during the test phase. It indicates that the technique can effectively self-adapt in a partial knowledge environment.

6.1.2 Coordination Overhead Reduction

The IDMG-based algorithm was aimed towards learning states where coordination is needed. In states where global goals are always satisfied, coordination is not required. Individual Q-learning is executed in these states to learn to optimize local goals only. Coordination is considered in states where global goals are violated (i.e., negative rewards are produced). As the number of coordinating states is reduced, the overall coordination overhead is also minimized.

The learning time steps of the technique are compared with the aforementioned three techniques. It was seen that the proposed technique converges faster due to the reduction of the coordination state space. The coordination overhead reduction also comes as a by-product from the partial knowledge incorporation. An issue to consider is the agent selection overhead introduced due to using a single peer agent. The agent to agent communication overhead was compared to the joint Q-learning method. It was seen that the average communication overhead is smaller. Theoretically, the overall coordination and communication overhead should be much smaller than the joint Q-learning because the proposed technique

coordinates in coordinating states only. This was also tested and as expected, the proposed technique was observed to have much lower total overhead in an episode.

6.1.3 Multi-Objective Self-Adaptation

Multi-Objective self-adaptation is considered to be a challenging problem in the literature [86]. The proposed multi-objective self-adaptation is simple and lightweight. The weighted summation of the reward functions is used in the IDMG-based model. The weights are dynamically adjusted in order to give more importance to the violated goals. A lemma is provided that helps to adjust the weight of these goals.

The aforementioned experiments are run on TAS with and without weight update. It was observed that the technique without weight update highly satisfied some goals where poorly performed for the others, resulting in goal violations. The proposed technique satisfied all the goals showing a stable performance.

6.2 Threats to Validity

The limitations of the experimental analysis are presented in this section entitled under internal, external and construct validity.

6.2.1 Internal Validity

This threat originates from the validity of the relationship between the independent and the dependent variables [87]. To limit this threat, the comparative studies are carefully performed. For example, the rewards of the proposed technique, independent Q-learning and joint Q-learning were compared to conclude that the introduction of partial knowledge and coordinating states cause higher rewards in earlier runs. The comparison with random action selection ensures that the randomness in the algorithm has not caused this result. The compari-

son with individual Q-learning confirms that the coordination with peer agent in the proposed method is effective. Finally, the comparison with joint Q-learning shows that using single agent and coordinating in specific states result in optimal behavior. Apart from these, each experiment is repeated many times to ensure its validity.

6.2.2 External Validity

It indicates how much the experiment generalizes to other systems. As decentralized self-adaptation has been applied widely, limiting the experiments to only two systems may induce this threat. To reduce this, well-known systems from the literature are chosen to conduct the experiments. The adaptive robot navigation is a popular toy problem that has been applied largely in many fields such as robotics, multiagent systems etc. [5, 7, 8, 9]. The Tele Assistance System is also a standard problem that resides in the Software Engineering for Self-Adaptive Systems (SEAMS) repository [83].

6.2.3 Construct Validity

Construct validity ensures that the measures used in a study are meaningful. Measuring the effectiveness and the performance of self-adaptive systems is still a research challenge. A very few works have been proposed such as [88, 89] etc. to evaluate such attributes. Nonetheless, the measurements in multiagent systems research are used to evaluate the proposed approach. This is because the approach is based on concepts from multiagent systems and its purposes are directly aligned with these systems. Thus, construct validity is tackled. To say from another angle, Fenton et al. mentioned that a metric is valid from the construct validity point of view if what it measures is consistent with our intuition [87]. In the experiments in this thesis, the average rewards, learning timesteps, communication time etc. are all intuitive metrics.

6.3 Future Work

The major challenge of decentralized self-adaptation assuming partial knowledge is addressed in this thesis. The following discussion describes some potential works in continuation of this research.

- Goal Dependency: The goals in a decentralized self-adaptive system may be dependent. For example, the satisfaction of one may be required for the satisfaction of the other or the satisfaction of one may cause the violation of another. There may be dependencies among local goals, global goals or local and global goals. In the future, the proposed technique will be extended by addressing these dependencies.

- Global Goal Distribution Overhead: The minimization and maximization goals are partitioned into local goals which may add some overhead to local goal optimization. Although the results show that convergence gets faster, reducing this overhead may improve the convergence speed.

- Engineering Principles: Engineering self-adaptive systems is still a research challenge [24, 90, 91]. The engineering of decentralized self-adaptive systems can be a challenging and exciting research direction. The proposed technique can be a framework directing the engineering approach. The thesis shows the fact that the partial knowledge incorporation should be an accepted principle during the engineering to ensure scalability. Concepts from distributed systems, self-adaptive systems, multiagent systems and feedback control-based system engineering can guide the researchers toward this.

- Formal Guarantees: Some works have been done on providing formal guarantees to decentralized self-adaptive systems. For example, DECIDE approach proposed by Calinescu et al. uses formal verification for this purpose. However, the decentralization technique in this work is based on global knowl-

edge. providing formal guarantees to a partial knowledge-based technique such as the proposed one is a complex task and out of the scope of this thesis. This will be addressed as a separate work in the future.

- Extension of Experimental Study: The experimental studies performed in this thesis will be conducted on other systems, specially industrial systems. How experimental studies should be performed for decentralized self-adaptive systems is not well-addressed in the literature. A defined way of validating such approaches is an area that needs further attention.

Appendix A

Testing Phase Actions Selections for Adaptive Robot Navigation

Table A.1: Testing Phase of Robot 1

State	Individual Q-Values					Selected Action	Peer Agent	Joint State	Peer Q-Values				Selected Action	Updated State
	Down	σ	Up	Left	Right				Down	Up	Left	Right		
-2 3	7.78	7.78	-	-	7.78	σ	Robot 2	-2 3 2 3	7.78	-	-	7.78	Down	-2 2
-2 2	8.09	8.09	7.48	7.48	8.09	Down								-2 1
-2 1	-	8.3	7.78	7.78	8.4	Right								-1 1
-1 1	8.71	102.54	8.09	8.09	8.71	σ	Robot 2	-1 1 1 1	8.71	8.09	8.09	8.71	Down	-1 0
-1 0	9.01	8.99	8.4	-	-	Down								-1 -1
-1 -1	8.76	9.31	8.71	8.71	9.31	Right								0 -1
0 -1	-	9.58	-	9.01	9.59	Right								1 -1
1 -1	9.83	103.74	9.31	9.31	9.83	σ	Robot 2	1 -1 -1 -1	9.83	9.31	9.31	9.83	Down	1 -2
1 -2	10.0	9.9	9.59	-	10.0	Down								1 -3
1 -3	-	9.96	9.83	-	10.0	Right								2 -3

Table A.2: Testing Phase of Robot 2

State	Individual Q-Values					Selected Action	Peer Agent	Joint State	Peer Q-Values				Selected Action	Updated State
	Down	σ	Up	Left	Right				Down	Up	Left	Right		
2 3	7.78	7.78	-	7.78	-	Left								1 3
1 3	8.09	8.08	-	-	7.48	Down								1 2
1 2	8.4	8.18	7.78	-	7.78	Down								1 1
1 1	8.68	102.47	8.09	8.71	8.09	σ	Robot 3	1 1 1 -1	8.69	8.09	8.71	8.09	Left	0 1
0 1	-	9.01	-	9.01	8.4	Left								-1 1
-1 1	9.31	102.93	8.71	8.76	8.71	σ	Robot 3	-1 1 1 1	9.31	8.71	8.76	8.71	Down	-1 0
-1 0	9.59	9.59	9.01	-	-	Down								-1 -1
-1 -1	9.83	104.14	9.31	9.83	9.31	σ	Robot 3	-1 -1 -1 1	9.83	9.31	9.83	9.31	Left	-2 -1
-2 -1	10.0	9.96	-	9.59	9.59	Down								-2 -2
-2 -2	10.0	9.86	9.83	9.83	9.83	Down								-2 -3

Table A.3: Testing Phase of Robot 3

State	Individual Q-Values					Selected Action	Peer Agent	Joint State	Peer Q-Values				Selected Action	Updated State
	Down	σ	Up	Left	Right				Down	Up	Left	Right		
2 -3	-	7.78	7.78	7.78	-	σ	Robot 4	2 -3 -2 -3	-	7.78	7.78	-	Up	2 -2
2 -2	7.48	8.09	8.09	8.09	7.48	Up								2 -1
2 -1	7.78	8.38	-	8.4	7.78	Left								1 -1
1 -1	8.09	100.0	8.71	8.7	8.09	σ	Robot 4	1 -1 -1 -1	8.09	8.71	8.7	8.09	Up	1 0
1 0	8.4	9.01	9.01	-	-	Up								1 1
1 1	8.73	98.28	8.76	9.31	8.71	σ	Robot 4	1 1 1 -1	8.7	8.76	9.31	8.71	Left	0 1
0 1	-	9.59	-	9.59	9.01	Left								-1 1
-1 1	9.31	103.61	9.83	9.83	9.31	σ	Robot 4	-1 1 1 1	9.31	9.83	9.83	9.31	Up	-1 2
-1 2	9.59	9.71	10.0	10.0	-	Left								-2 2
-2 2	9.83	9.86	10.0	9.83	9.83	Up								-2 3

Table A.4: Testing Phase of Robot 4

State	Individual Q-Values					Selected Action	Peer Agent	Joint State	Peer Q-Values				Selected Action	Updated State
	Down	σ	Up	Left	Right				Down	Up	Left	Right		
-2 -3	-	7.78	7.78	-	7.78	Right								-1 -3
-1 -3	-	8.08	8.09	7.48	-	Up								-1 -2
-1 -2	7.78	8.38	8.39	7.78	-	Up								-1 -1
-1 -1	8.08	96.14	8.6	8.08	8.71	σ	Robot 1	-1 -1 -1 0	8.08	8.13	8.07	8.69	Right	0 -1
0 -1	-	8.87	-	8.39	9.01	Right								1 -1
1 -1	8.7	92.91	9.31	8.8	8.76	σ	Robot 1	1 -1 0 -1	8.7	9.3	8.68	8.76	Up	1 0
1 0	9.01	9.58	9.59	-	-	Up								1 1
1 1	9.31	103.29	9.83	9.31	9.83	σ	Robot 1	1 1 1 -2	9.31	9.83	9.3	9.83	Right	2 1
2 1	-	9.88	10.0	9.59	9.59	Up								2 2
2 2	9.83	9.92	10.0	9.83	9.83	Up								2 3

References

[1] Danny Weyns, Bradley R. Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M. Göschka. On patterns for decentralized control in self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, pages 76–107, 2010.

[2] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger M. Kienle, Marin Litoiu, Hausi A. Müller, Mauro Pezzè, and Mary Shaw. Engineering self-adaptive systems through feedback loops. In *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*, pages 48–70, 2009.

[3] Giovanna Di Marzo Serugendo, Marie Pierre Gleizes, and Anthony Karageorgos. Self-organisation and emergence in MAS: an overview. *Informatica (Slovenia)*, 30(1):45–54, 2006.

[4] Danny Weyns, Sam Malek, and Jesper Andersson. On decentralized self-adaptation: lessons from the trenches and challenges for the future. In *2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2010, Cape Town, South Africa, May 3-4, 2010*, pages 84–93, 2010.

[5] Matthijs T. J. Spaan and Francisco S. Melo. Interaction-driven markov games for decentralized multiagent planning under uncertainty. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, 2008, Volume 1*, pages 525–532, 2008.

[6] Danny Weyns and Radu Calinescu. Tele assistance system (tas). `https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/tas/`.

[7] Francisco S. Melo and Manuela M. Veloso. Learning of coordination: exploiting sparse interactions in multiagent systems. In *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May 10-15, 2009, Volume 2*, pages 773–780, 2009.

[8] Yi Guo and Lynne E. Parker. A distributed and optimal motion planning approach for multiple mobile robots. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation, ICRA 2002, May 11-15, 2002, Washington, DC, USA*, pages 2612–2619, 2002.

[9] Hans Jacob S. Feder, John J. Leonard, and Christopher M. Smith. Adaptive mobile robot navigation and mapping. *I. J. Robotics Res.*, 18(7):650–668, 1999.

[10] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley R. Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 37(10):46–54, 2004.

[11] Autonomic Computing et al. An architectural blueprint for autonomic computing. *IBM White Paper*, 2006.

[12] Naeem Esfahani, Ahmed M. Elkhodary, and Sam Malek. A learning-based framework for engineering feature-oriented self-adaptive software systems. *IEEE Trans. Software Eng.*, 39(11):1467–1493, 2013.

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

[14] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley R. Schmerl, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, Danny Weyns, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ronald J. Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl M. Göschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi, Gabor Karsai, Jeff Kramer, Antónia Lopes, Jeff Magee, Sam Malek, Serge Mankovski, Raffaela Mirandola, John Mylopoulos, Oscar Nierstrasz, Mauro Pezzè, Christian Prehofer, Wilhelm Schäfer, Richard D. Schlichting, Dennis B. Smith, João Pedro Sousa, Ladan Tahvildari, Kenny Wong, and Jochen Wuttke. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, pages 1–32, 2010.

[15] Vivek Nallur and Rami Bahsoon. A decentralized self-adaptation mechanism for service-based applications in the cloud. *IEEE Trans. Software Eng.*, 39(5):591–612, 2013.

[16] Daniel Sykes, Jeff Magee, and Jeff Kramer. Flashmob: distributed adaptive self-assembly. In *2011 ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2011, Waikiki, Honolulu , HI, USA, May 23-24, 2011*, pages 100–109, 2011.

[17] Radu Calinescu, Simos Gerasimou, and Alec Banks. Self-adaptive software with decentralised control loops. In *Fundamental Approaches to Software Engineering - 18th International Conference, FASE 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 235–251, 2015.

[18] Jim Dowling, Raymond Cunningham, Eoin Curran, and Vinny Cahill. Building autonomic systems using collaborative reinforcement learning. *Knowledge Eng. Review*, 21(3):231–238, 2006.

[19] Hongbing Wang, Qin Wu, Xin Chen, Qi Yu, Zibin Zheng, and Athman Bouguettaya. Adaptive and dynamic service composition via multi-agent reinforcement learning. In *2014 IEEE International Conference on Web Services, ICWS, 2014, Anchorage, AK, USA, June 27 - July 2, 2014*, pages 447–454, 2014.

[20] Mauro Caporuscio, Mirko D'Angelo, Vincenzo Grassi, and Raffaela Mirandola. Reinforcement learning techniques for decentralized self-adaptive service assembly. In *Service-Oriented and Cloud Computing - 5th IFIP WG 2.14 European Conference, ESOCC 2016, Vienna, Austria, September 5-7, 2016, Proceedings*, pages 53–68, 2016.

[21] Vincenzo Grassi, Moreno Marzolla, and Raffaela Mirandola. Qos-aware fully decentralized service assembly. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2013, San Francisco, CA, USA, May 20-21, 2013*, pages 53–62, 2013.

[22] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pages 746–752, 1998.

[23] Miguel Jiménez, Mario Piattini, and Aurora Vizcaíno. Challenges and improvements in distributed software development: A systematic review. *Adv. Software Engineering*, 2009:710971:1–710971:14, 2009.

[24] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*, pages 1–26, 2009.

[25] Jesper Andersson, Rogério de Lemos, Sam Malek, and Danny Weyns. Modeling dimensions of self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*, pages 27–47, 2009.

[26] Paul Horn. Autonomic computing: Ibm\'s perspective on the state of information technology. 2001.

[27] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch-Lafuente, and Andrea Vandin. A conceptual framework for adaptation. In *Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, pages 240–254, 2012.

[28] Jeff Kramer and Jeff Magee. Self-managed systems: an architectural challenge. In *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA*, pages 259–268, 2007.

[29] Erann Gat et al. On three-layer architectures. *Artificial intelligence and mobile robots*, 195:210, 1998.

[30] Simon Dobson, Spyros G. Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *TAAS*, 1(2):223–259, 2006.

[31] Elisabetta Di Nitto, Daniel J. Dubois, and Raffaela Mirandola. On exploiting decentralized bio-inspired self-organization algorithms to develop real systems. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2009, Vancouver, BC, Canada, May 18-19, 2009*, pages 68–75, 2009.

[32] Eduardo Camponogara, Dong Jia, Bruce H Krogh, and Sarosh Talukdar. Distributed model predictive control. *IEEE Control Systems*, 22(1):44–52, 2002.

[33] Ivana Dusparic and Vinny Cahill. Distributed w-learning: Multi-policy optimization in self-organizing systems. In *Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2009, San Francisco, California, USA, September 14-18, 2009*, pages 20–29, 2009.

[34] Horst F. Wedde and Sebastian Senge. Beejama: A distributed, self-adaptive vehicle routing guidance approach. *IEEE Trans. Intelligent Transportation Systems*, 14(4):1882–1895, 2013.

[35] Sandip Sen and Gerhard Weiss. Learning in multiagent systems. *Multiagent systems: A modern approach to distributed artificial intelligence*, pages 259–298, 1999.

[36] Daniel Sykes, William Heaven, Jeff Magee, and Jeff Kramer. From goals to components: a combined approach to self-management. In *2008 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2008, Leipzig, Germany, May 12-13, 2008*, pages 1–8, 2008.

[37] Chih-Han Yu and Radhika Nagpal. A self-adaptive framework for modular robots in a dynamic environment: Theory and applications. *I. J. Robotics Res.*, 30(8):1015–1036, 2011.

[38] Guillaume Gauvrit, Erwan Daubert, and Françoise André. SAFDIS: A framework to bring self-adaptability to service-based distributed applications. In *36th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2010, Lille, France, September 1-3, 2010*, pages 211–218, 2010.

[39] Jim Dowling, Eoin Curran, Raymond Cunningham, and Vinny Cahill. Using feedback in collaborative reinforcement learning to adaptively optimize MANET routing. *IEEE Trans. Systems, Man, and Cybernetics, Part A*, 35(3):360–372, 2005.

[40] Kenji Leibnitz, Naoki Wakamiya, and Masayuki Murata. Biologically inspired self-adaptive multi-path routing in overlay networks. *Commun. ACM*, 49(3):62–67, 2006.

[41] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark Harrison, Markus Eisenhauer, et al. Internet of things strategic research roadmap. *Internet of Things-Global Technological and Societal Trends*, 1(2011):9–52, 2011.

[42] Nathalia Moraes do Nascimento and Carlos José Pereira de Lucena. Fiot: An agent-based framework for self-adaptive and self-organizing applications based on the internet of things. *Inf. Sci.*, 378:161–176, 2017.

[43] Danny Weyns. Engineering Smart Cyber Physical Systems. `https://www.youtube.com/watch?v=jjvBnvA8GzA`, October 2015.

[44] Pieter Vromant, Danny Weyns, Sam Malek, and Jesper Andersson. On interacting control loops in self-adaptive systems. In *2011 ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2011, Waikiki, Honolulu , HI, USA, May 23-24, 2011*, pages 202–207, 2011.

[45] Vivek Nallur, Nicolás Cardozo, and Siobhán Clarke. Clonal plasticity: a method for decentralized adaptation in multi-agent systems. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2016, Austin, Texas, USA, May 14-22, 2016*, pages 122–128, 2016.

[46] Ignacio Rojas, Jesús González, Héctor Pomares, Juan J. Merelo Guervós, Pedro Ángel Castillo Valdivieso, and Gustavo Romero. Statistical analysis of the main parameters involved in the design of a genetic algorithm. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 32(1):31–37, 2002.

[47] Mohamed Bakhouya and Jaafar Gaber. Bio-inspired approaches for engineering adaptive systems. In *Proceedings of the 5th International Conference on Ambient Systems, Networks and Technologies (ANT 2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014), Hasselt, Belgium, June 2-5, 2014*, pages 862–869, 2014.

[48] Marco Dorigo and Mauro Birattari. Ant colony optimization. In *Encyclopedia of Machine Learning*, pages 36–39. 2010.

[49] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[50] Zhenyu Yang, Ke Tang, and Xin Yao. Large scale evolutionary optimization using cooperative coevolution. *Inf. Sci.*, 178(15):2985–2999, 2008.

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

[52] John F Nash et al. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[53] Ann Nowé, Peter Vrancx, and Yann-Michaël De Hauwere. Game theory and multi-agent reinforcement learning. In *Reinforcement Learning*, pages 441–470. Springer, 2012.

[54] Junling Hu and Michael P. Wellman. Nash q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

[55] Kishan Kumar Ganguly and Kazi Sakib. Decentralization of control loop for self-adaptive software through reinforcement learning. In *Software Engineering Conference Workshops (APSECW), 2017 24th Asia-Pacific*, pages 134–141. IEEE, 2017.

[56] Theodore J. Lambert III, Marina A. Epelman, and Robert L. Smith. A fictitious play approach to large-scale optimization. *Operations Research*, 53(3):477–489, 2005.

[57] Sankardas Roy, Charles Ellis, Sajjan G. Shiva, Dipankar Dasgupta, Vivek Shandilya, and Qishi Wu. A survey of game theory as applied to network security. In *43rd Hawaii International International Conference on Systems Science (HICSS-43 2010), Proceedings, 5-8 January 2010, Koloa, Kauai, HI, USA*, pages 1–10, 2010.

[58] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[59] Boris Pittel. On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, 1987.

[60] Spyros Voulgaris et al. *Epidemic-based self-organization in peer-to-peer systems*. PhD thesis, VU University Amsterdam, 2006.

[61] Davide Devescovi, Elisabetta Di Nitto, Daniel J. Dubois, and Raffaela Mirandola. Self-organization algorithms for autonomic systems in the selflet approach. In *Proceedings of the 1st International Conference on Autonomic Computing and Communication Systems, Autonomics 2007, 28-30 October 2007, Rome, Italy*, page 26, 2007.

[62] Gerald Tesauro, David M. Chess, William E. Walsh, Rajarshi Das, Alla Segal, Ian Whalley, Jeffrey O. Kephart, and Steve R. White. A multi-agent systems approach to autonomic computing. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*, pages 464–471, 2004.

[63] Wolfgang Pree. *Design patterns for object-oriented software development.* ACM Press books. Addison-Wesley, 1994.

[64] Shang-Wen Cheng, David Garlan, and Bradley R. Schmerl. Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems, SEAMS 2006, Shanghai, China, May 21-22, 2006*, pages 2–8, 2006.

[65] Peyman Oreizy, Michael M Gorlick, Richard N Taylor, Dennis Heimhigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S Rosenblum, and Alexander L Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and Their Applications*, 14(3):54–62, 1999.

[66] Danny Weyns and Tanvir Ahmad. Claims and evidence for architecture-based self-adaptation: A systematic literature review. In *Software Architecture - 7th European Conference, ECSA 2013, Montpellier, France, July 1-5, 2013. Proceedings*, pages 249–265, 2013.

[67] Ioannis Georgiadis, Jeff Magee, and Jeff Kramer. Self-organising software architectures for distributed systems. In *Proceedings of the First Workshop on Self-Healing Systems, WOSS 2002, Charleston, South Carolina, USA, November 18-19, 2002*, pages 33–38, 2002.

[68] KTeam. Koala 2.5. `https://www.k-team.com/mobile-robotics-products/koala`, 2017.

[69] Alberto Montresor and Márk Jelasity. Peersim: A scalable P2P simulator. In *Proceedings P2P 2009, Ninth International Conference on Peer-to-Peer Computing, 9-11 September 2009, Seattle, Washington, USA*, pages 99–100, 2009.

[70] Roy Sterritt and Michael G. Hinchey. Biologically-inspired concepts for self-management of complexity. In *11th International Conference on Engineering of Complex Computer Systems (ICECCS 2006), 15-17 August 2006, Stanford, California, USA*, pages 163–168, 2006.

[71] Amy D Bradshaw. Evolutionary significance of phenotypic plasticity in plants. In *Advances in genetics*, volume 13, pages 115–155. Elsevier, 1965.

[72] John Conway. The game of life. *Scientific American*, 223(4):4, 1970.

[73] Luca Florio. Decentralized self-adaptation in large-scale distributed systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, pages 1022–1025, 2015.

[74] Mark Humphreys. *Action selection methods using reinforcement learning.* PhD thesis, University of Cambridge, UK, 1997.

[75] Vinny Reynolds, Vinny Cahill, and Aline Senart. Requirements for an ubiquitous computing simulation and emulation environment. In *Proceedings of the First International Conference in Integrated Internet Ad Hoc and Sensor Networks, InterSense 2006, Nice, France, May 30-31, 2006*, page 1, 2006.

[76] Eyhab Al-Masri and Qusay H. Mahmoud. Qos-based discovery and ranking of web services. In *Proceedings of the 16th International Conference on Computer Communications and Networks, IEEE ICCCN 2007, Turtle Bay Resort, Honolulu, Hawaii, USA, August 13-16, 2007*, pages 529–534, 2007.

[77] Kishan Kumar Ganguly and Kazi Sakib. A reusable adaptation component design for learning-based self-adaptive systems. *ICSEA 2017*, page 255, 2017.

[78] Michel Tokic and Günther Palm. Value-difference based exploration: Adaptive control between epsilon-greedy and softmax. In *KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI, Berlin, Germany, October 4-7,2011. Proceedings*, pages 335–346, 2011.

[79] Josef Hofbauer and William H Sandholm. On the global convergence of stochastic fictitious play. *Econometrica*, 70(6):2265–2294, 2002.

[80] Stuart J. Russell and Peter Norvig. *Artificial intelligence - a modern approach, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, 2003.

[81] Yann-Michaël De Hauwere, Peter Vrancx, and Ann Nowé. Learning multi-agent state space representations. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*, pages 715–722, 2010.

[82] A foundation for engineering decentralized self-adaptive software systems - federates. `http://homepage.lnu.se/staff/daweaa/projects/FEDERATES.htm`.

[83] Software engineering for self-adaptive systems - tele assistance system (tas). `https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/tas/`.

[84] Danny Weyns and Radu Calinescu. Tele assistance: A self-adaptive service-based system exemplar. In *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18-19, 2015*, pages 88–92, 2015.

[85] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[86] Antonio Filieri, Henry Hoffmann, and Martina Maggio. Automated multi-objective control for self-adaptive software design. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, pages 13–24, 2015.

[87] Norman E. Fenton and Shari Lawrence Pfleeger. *Software metrics - a practical and rigorous approach (2. ed.)*. International Thomson, 1996.

[88] Norha M. Villegas, Hausi A. Müller, Gabriel Tamura, Laurence Duchien, and Rubby Casallas. A framework for evaluating quality-driven self-adaptive software systems. In *2011 ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2011, Waikiki, Honolulu , HI, USA, May 23-24, 2011*, pages 80–89, 2011.

[89] Shang-Wen Cheng, David Garlan, and Bradley R. Schmerl. Evaluating the effectiveness of the rainbow self-adaptive system. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2009, Vancouver, BC, Canada, May 18-19, 2009*, pages 132–141, 2009.

[90] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17:184–206, 2015.

[91] Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.