UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

Decentralised Structural Adaptation for Agent Organisations

by Ramachandra Kota

Supervisors: Professor Nicholas R. Jennings and Dr. Nicholas Gibbins
Examiner: Dr. Enrico H. Gerding

A mini-thesis submitted for transfer from MPhil to PhD

February 25, 2008

ABSTRACT

Autonomic systems, capable of self-management, are being advocated as a solution to the problem of maintaining modern, large, complex computing systems. Given this, we believe self-organising multi-agent systems provide a convenient paradigm to develop these autonomic systems because such self-organising systems can arrange and re-arrange their structure autonomously, without any external control, in order to adapt to changing requirements and environmental conditions. Furthermore, such systems need to be decentralised, so that they are robust against failures; again, this characteristic fits with the multi-agent paradigm. With this motivation, this report explores the area of self-organisation in agent systems, and particularly focuses on the decentralised structural adaptation of agent organisations.

In more detail, self-organisation has been generated in agent systems using various approaches like stigmergy, reinforcement mechanisms, cooperative actions of agents and reward based mechanisms for selfish agents. However, none of these are directly applicable to agent organisations because they cannot be incorporated into deliberative agents with fixed properties, working towards organisational goals. Here, we particularly focus on such problem solving agent organisations because they provide a suitable representation for autonomic systems. Against this background, we investigate and develop mechanisms to incorporate decentralised structural adaptation in organisations that seek to improve their performance.

More specifically still, we provide a simple model for representing problem solving agent organisations and an evaluation mechanism to determine the performance of the organisation for any given set of tasks. This serves as the framework within which we investigate approaches for developing reorganisation. Furthermore, we present a structural adaptation method that enables the agents to modify the organisational structure based on their history of interactions with other agents, in order to improve the performance of the organisation. Finally, our empirical evaluation shows that our method is successful in improving the organisation performance and is comparable to that managed by an external infinitely resourceful central agent.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

As computing systems get ever larger and more complex, it is becoming correspondingly harder to maintain them due to their increasing size and the interconnectedness of their numerous components. Given this fact, several researchers suggest tackling this problem by making them *autonomic*, so that they manage themselves (Kephart and Chess, 2003; Mainsah, 2002). In more detail, autonomic systems exhibit the property of self-management which involves autonomously adjusting their operations according to changing conditions. Also, these autonomic systems are expected to be decentralised so that they are robust without any single point of failure. Against this background, Tesauro et al. (2004) argue that such systems can naturally be developed by adopting a multi-agent systems approach as agents are autonomous and provide a suitable paradigm for developing decentralised systems. This, in turn, leads to the area of developing multi-agent systems that exhibit self-management. Now, the area of research that deals with self-management in agent systems is *self-organisation*, *the mechanism or the process enabling the system to change its organisation without explicit external command during its execution time* (Di Marzo Serugendo et al., 2005b). Thus, self-organisation can consist of either forming an organisation from disordered entities or reorganising an existing organisation, similar to maintaining computing systems. Therefore, to further aid the development of autonomic systems, our work explores the area of self-organisation in agent systems.

In more detail, self-organisation can be generated in multi-agent systems in several ways, as demonstrated by Di Marzo Serugendo et al. (2006) and Bernon et al. (2006). For example, self-organisation may emerge from stigmergic or reinforcement mechanisms in agents (Mano et al., 2006) or can arise from the locally cooperative actions of the agents (Kamboj and Decker, 2006; Capera et al., 2003a). Even virtual organisations (Norman et al., 2004) can be considered to be an example of self organisation among agents acting in a competitive scenario as the agents autonomously organise and reorganise themselves without any external intervention. However, most of the work on self-organisation in agents deals with multi-agent systems that do not have any organisation defined explicitly; the social interactions of the agents are not guided by definite regulations. We require that our design of agent systems can be mapped onto computing systems that perform tasks, so that our research on self-organising agents is useful for the development of autonomic systems which are composed of such computing systems. To this end, we focus specifically on developing self-organisation techniques for multi agent systems that act as a problem-solving organisation (organisations that receive inputs, perform tasks and return results)[1]. Our study of the various self-organisation mechanisms reveals that most of them are not applicable to an explicitly modelled agent organisation because they cannot be incorporated into agents that are working towards organisational goals (see Section 2.2.3 for more details). The few self-organisation mechanisms that do consider agent organisations are centralised in nature, thereby not addressing the requirement for decentralisation. A few other self-organisation techniques that can be employed in agent organisations work by modifying the agents themselves. However, such an approach is not viable for developing autonomic systems as changing the internal configurations of the computing systems is not always possible. Moreover, we also seek to ensure that our methods are maximally applicable.

Against this background, we seek to develop a decentralised reorganisation method that can be employed by the agents in an organisation to either maintain or improve the performance of that organisation. Following the self-organisation principles, the method should be a continuous process that is followed by all the agents utilising only their local information for the benefit of the organisation as a whole. Furthermore, we attempt to develop adaptation techniques that can be applied to organisations without having to change the agents or their internal characteristics; we are primarily interested in reorganisation based on the structural adaptation of the organisation. Thus, our reorganisation mechanism will serve as a self-management tool similar to those visualised in autonomic systems. However, before undertaking this task, two preliminary issues need to be resolved — we need to have a simple but explicit model of a problem-solving agent organisation that can act as the platform on which to found our structural adaptation mechanism, and we also need an evaluation procedure that measures the performance of the organisation thereby enabling us to verify the effectiveness of the resultant adaptation. We further elucidate our objectives in the next section.

[1]This is in contrast to organisations which just provide guidelines to be obeyed by agents participating in the organisation to achieve their individual goals (Sierra et al., 2004). These organisations do not have any specific goals to achieve but only act as regulating authorities and so are not relevant in this context.

1.1 Research Objectives

We specify the objectives of this research in terms of the following requirements:

1. **Formulate a model of problem-solving agent organisations**

   The purpose of the organisation model is to serve as a framework on which the adaptation mechanism can be developed. For this reason, we do not wish to develop a highly sophisticated model since this is not our main focus. However, it needs to have the general organisational characteristics like members, relationships and authority structures, all of which can be modelled in a straightforward manner at this time. To further focus our research, we seek to develop a model for *problem-solving agent organisations*. By this, we refer to those agent organisations that receive inputs (as tasks), perform actions on the basis of these inputs (task processing or execution) and return results, thereby solving a problem.

2. **Develop an evaluation mechanism that measures the performance of an agent organisation.**

   The performance of an organisation is measured in terms of how well it performs its tasks. Towards this end, we aim to develop mathematical functions that calculate the efficiency of an organisation executing a given set of tasks. Such an evaluation mechanism is necessary because it will act as a criterion for determining the effectiveness of the adaptation methods.

3. **Develop a completely decentralised structural adaptation method that can be employed by the agents in a problem-solving agent organisation to improve the performance of the organisation as a whole.**

   The adaptation method needs to be agent-based and be employable by any agent, at any level of the organisation, and at any time, on the basis of its local views. However, the resultant reorganisation of the structure should benefit the whole organisation and not just the particular agents that initiate the reorganisation. We lay emphasis on the decentralised nature of the method so that it is robust without any critical points of failure. Also, since the purpose of this research is guided by the concept of autonomic systems (as discussed earlier), we focus on adaptation methods that target the organisation characteristics like the structure rather than the individual agent characteristics.

4. **Evaluate the effectiveness of this structural adaptation mechanism.**

The efficacy of the adaptation technique will be measured by comparing the performance of the organisations using this method against those not reorganising and those reorganising using other approaches. Such an analysis is necessary to both validate our research and also discover avenues for future work.

By achieving these objectives, we claim to advance the state of the art in the domain of problem-solving agent organisations in the following ways:

- Developing the first completely decentralised adaptation method for problem-solving agent organisations placed in dynamic task environments that works solely by modifying the structural aspects of the organisation, without changing the agents or their internal characteristics.
- Developing the first decentralised structural adaptation method that focuses on the efficiency of problem-solving agent organisations.

In the present report, we partially achieve these objectives by developing the adaptation method for a limited model of problem solving agent organisations in which the organisation is assumed to be *closed* (no agents can leave nor can any new agents join the organisation), the agents are assumed to be *invariant* (the properties of the agents are fixed) and the environmental conditions (communication costs and so on, but not the tasks) are assumed to be unchanging. This has resulted in the following paper:

Ramachandra Kota, Nicholas Gibbins and Nicholas R. Jennings (2008). Decentralised structural adaptation in agent organisations. Submitted to *International Workshop on Organised Adaptation in Multi-Agent Systems (at AAMAS'08).*

Next, we describe the structure of this report by highlighting the contents of the following chapters.

1.2 Report Structure

The next chapter discusses related work in the areas of interest, which are organisation modelling, organisation performance evaluation, self-organisation and other reorganisation mechanisms.

Chapter 3 presents the framework of agent organisation that is developed and used in this work. We also introduce and explain an evaluation mechanism to measure the efficiency of the organisation.

Chapter 4 details the constraints that have been placed on the organisation model for developing an adaptation mechanism and then goes on to present a decentralised structural adaptation method which enables the agents to modify the organisation structure on the basis of their history of interactions.

Chapter 5 records the experiments conducted on the adaptation method and analyses the obtained results.

Finally, Chapter 6 discusses how the work, presented in this report, will be extended in the future and presents a time-line for this research until the end of the thesis.

Chapter 2

Related Work

This chapter presents an overview of current research in the areas of organisation modelling and self-organisation in multi-agent systems. The first section studies the main work in modelling agent organisations and compares it with respect to the first and second requirements stated in Section 1.1 which refer to developing a model for problem-solving agent organisations and an evaluation mechanism to measure its performance. The next section presents the motivation and characteristics of self-organisation in multi-agent systems and follows them up with a description and analysis of the various techniques developed in this particular field. The third and final section summarises the chapter by highlighting the work that provides the point of departure for this study and also drawing attention to the open issues that need to be addressed to meet the requirements laid out in Section 1.1.

2.1 Modelling of Agent Organisations

Organisation modelling involves modelling the agents comprising the organisation, the organisational characteristics (structure and norms) and the task environment (detailing the tasks that should be performed by the organisation). The following subsections discuss various ways in which an organisation can be modelled in multi-agent systems by considering each of the three components (tasks, organisation characteristics and agents) individually. This section concludes with a discussion of the methods for evaluating the performance of agent organisations. This discussion is important in this context, because our second requirement (Section1.1) specifies designing an evaluation mechanism applicable for agent organisations.

2.1.1 Modelling Tasks

Typically, agent organisations execute some task(s). Therefore, the tasks of an organisation form an integral part of its description. Specifically, the set of tasks can be considered to be the problem space of the organisation (Carley and Gasser, 1999). Hence, developing a representation model for the tasks is a necessary step in the process of modelling an organisation.

In more detail, the tasks can be atomic or made up of two or more tasks (or subtasks) which, in turn, may be composed of other tasks. The tasks may have dependencies among them, resulting in a temporal ordering of the tasks in the organisation. In this context, Thompson (1967) identifies three kinds of such dependencies — pooled, sequential and reciprocal. Two or more tasks whose results are jointly required to execute another task are said to be in a pooled dependency relation with each other. A sequential dependency exists between tasks if they have to be performed in a particular sequence. Finally, a reciprocal dependency exists if the tasks are mutually dependent on each other and have to be executed at the same time. However, the tasks dependencies as suggested by Thompson have subsequently been interpreted in different ways in different models.

In particular, Jin and Levitt (1996) model the task dependencies in their 'Virtual Design Team (VDT)' closely following Thompson's model. However, they extend the sequential dependency by representing it as a *successor relationship* between the tasks and further classifying it as a finish-to-start successor (if the task can only be started on the completion of the other task comprising the dependency) or a start-to-start successor (if the task can be started after the start of the other task). Similarly, they consider two types of reciprocal dependencies — information-related and work-related. The former is present between two tasks when some information from the execution of one task is required by another and vice versa. The latter is present in those cases where a change in the execution of one task effects the other and vice versa. This representation is particularly useful if the dependencies are to be modelled in detail.

In contrast, in the PCANS model, Krackhardt and Carley (1998) demonstrate that both pooled and reciprocal dependencies, as described by Thompson, can be derived from sequential dependencies. For example, if task 1 is dependent on the completion of tasks 2 and 3, then tasks 2 and 3 share a pooled dependency relationship. But that also means that task 1 is sequentially dependent on task 2 and task 3. Thus, all the tasks that form the sequential dependencies of a particular task are in a pooled dependency. Similarly, two tasks sharing a reciprocal dependency with each other can be broken into smaller tasks which have a series of sequential dependencies. For example, let there be two tasks, 1 and 2, having a reciprocal dependency with each other. They can be divided as, say, task 1a and task 1b (representing the first task) and task 2a and task 2b (representing the second one) such that task 1a is sequentially dependent on task 2a, which is sequentially dependent on task 1b, which, in turn, is sequentially dependent on task 2b. Thus, their representation enables the designer to model just a single dependency.

Other than the task dependencies, the task environment of an organisation can be further characterised on the basis of the degree of repetition, volatility, bias, and complexity (Carley and Gasser, 1999). Thus, tasks could be repetitive, quasi-repetitive (same type of tasks, but some details in the specific instances are different) or non-repetitive. Volatility denotes the rate of change of the tasks. Bias represents the extent to which all possible tasks may have the same outcome, while complexity denotes the amount of processing required by the tasks. Changes in the task environment can also be classified as — sudden change, oscillating, and gradual change. In sudden change, the task environment changes considerably in a short span of time. Oscillating environments are those in which the tasks keep fluctuating between two or more types at fairly regular intervals of time. In gradually changing environments, the change in tasks is uniformly distributed over a considerable period of time.

For our present requirements, we just require a simple task model containing dependencies, and hence we will use the PCANS model to represent our tasks. Also, the characteristics of the task environment, as described above, will be considered while generating the input tasks during experimentation (see Section 5.1.1).

2.1.2 Modelling Organisational Characteristics

Approaches towards organisational design in multi-agent systems can be considered to be either agent-centric or organisation-centric (Lematre and Excelente, 1998). The former focus on the social characteristics of agents like joint intentions, social commitment, collective goals and so on. Therefore, the organisation is a result of the social behaviour of the agents and is not created explicitly by the designer. On the other hand, in organisation-centric approaches, the focus of design is the organisation which has some rules or norms which the agents must follow. Thus, the organisational characteristics are *imposed* on the agents. Dignum and Dignum (2005) show that an explicit organisation structure helps in the achievement of the objectives of the organisation as these goals may be wider than an individual agent can perceive. Due to these reasons, and also because our requirements relate primarily to problem-solving agent organisations (see Section 1.1), we only study organisations in multi-agent systems whose design is modelled explicitly. This means we exclude those approaches based on agent-centric organisation design (Gasser et al., 1988; Cohen and Levesque, 1991).

Against this background, several models for depicting computational organisations have been developed by researchers in the multi-agent systems community. We shall examine the main ones in the rest of this subsection:

- **Opera:** The OperA framework (Dignum, 2003) is useful for formal specification of agent societies. Its organisational model specifies the organisational characteristics on the basis of the social structure (role characteristics including skills and relations), interaction structure (agent interactions constitute scenes), normative structure (describing expectations and boundaries for agent behaviour) and communication structure (ontology for knowledge representation and communication language). This work is then extended by the OMNI framework (Vazquez-Salceda et al., 2005) which enables the designer to specify the organisational structure, the interactions between the agents and the normative structure independent of the design of the agents. So, it provides a much broader framework for designing agent organisations. However, in both of these frameworks, the agents are not permitted to modify the organisational characteristics that have been pre-designed and, hence, they do not provide a suitable platform for reorganisation and do not meet our requirements.

- **Islander:** Islander (Sierra et al., 2004) uses the following elements to model the organisation — dialogic framework, scenes, performative structure and norms. The dialogic framework defines the roles that can be adopted by the agents. Every role defines a fixed pattern of behaviour expected from the agent playing that role. The dialogic framework also defines the relationships between the roles and the communication ontology. The activities in the organisation are called scenes and involve instances of interactions between the agents playing the roles. A collection of related scenes form a performative structure. The norms represent the commitments, obligations and rights of the participating agents. All these are defined during design time and cannot be changed during execution. Hence, this model too is not flexible enough to incorporate reorganisation.

- **AGR:** A simpler model is provided by Ferber and Gutknecht (1998) and later extended by Ferber et al. (2003). They define an organisation as *a structural relationship between a collection of agents*. Specifically, a meta-model is presented to describe organisations based on agents, groups and roles (AGR). Agents are part of one or more groups and play specific roles within the groups. These groups are created by the agents and the creating agent assumes the group manager role. The structure of the group identifies all the roles and interactions that can appear in the group. The group structure consists of a tuple containing all the possible roles, an interaction graph specifying all valid interactions between the roles in the group and an interaction language that should be followed by the group. The organisation structure contains the set of groups and the possible interactions between the roles belonging to different groups. Therefore this model provides a reasonable representation for an organisation containing several groups or departments. While, we do not require several groups in our organisation, we will use the idea of the organisation structure determining the interactions between the members as this interpretation emphasises the importance of the structure.

- **Moise:** A somewhat similar approach is followed by Moise (Hannoun et al., 2000), which considers an organisation structure as a graph defined by a set of roles, a set of different types of links and a set of groups. A role consists of a set of missions. Here, a mission represents a permitted behaviour in the system and is defined by a set of goals, plans, actions and resources. An agent playing a role must obey some permitted behaviours which are specified by the missions comprising the role. The missions may be viewed as the set of services that should be provided by the agent playing the role. Organisation links are arcs between the roles and represent the interactions between the roles. The model suggests three types of links — communication, authority and acquaintance. Communication links specify the kind of communication that can exist between the roles, the protocols to be followed and the particular missions for which they can be used. Authority links represent the subordination of one role to another along with the context for which it is valid. The context is defined by the missions associated with the link. The acquaintance links of a role specify all the roles about which the agent playing this role can possess information and use in its decision mechanism. The definition of groups contains a set of roles, a set of missions (which is a subset of the combined set of missions of the roles belonging to the group) and a set of links which exist between roles belonging to the group. Some of the ideas used in this model, especially those relating to the organisation structure will be used in the work on developing our model of organisation. These ideas of relations and interactions with their corresponding graphs provide a good insight into the influence of a structure on the organisation's performance and is therefore helpful to us.

- **VDT:** A slightly different approach is followed by the Virtual Design Team (VDT) framework (see Section 2.1.1). Its purpose is to develop a computational model of real-life project organisations (an organisation involved in completion of a project or set of tasks). It does not use the agent-role paradigm. Instead, the agents are fixed to their duties and are called actors. The organisation structure is composed of two structures — a control structure and a communication structure. The former determines the supervision and authority between the members and specifies who reports to whom, while the latter specifies who can talk to whom. An additional parameter representing the *formalisation* of the organisation determines the frequency of communication between the actors. Evidently, VDT attempts to model a problem-solving organisation, and therefore, very relevant for our requirements. However, it lacks flexibility in the organisation structure, as it only permits purely hierarchical organisations. Therefore, we do not directly use the whole VDT model but only some parts of it.

- **ODML and KB-ORG:** More recently, ODML (Horling and Lesser, 2008) was developed as a quantitative framework for representing organisations. It uses a mathematical syntax, rather than the commonly used structures and norms representation for denoting organisations. The organisational models produced by ODML can be quantitatively compared against each other for a given set of requirements. Therefore, a search space of organisation instances can be explored to arrive at the most befitting design. However, as noted by the authors themselves, a significant amount of domain knowledge and effort is required to build the models. Another work on similar lines is KB-ORG (Sims et al., 2008), which is an automated knowledge based process for designing organisations. KB-ORG searches and prunes the design search space by using a search algorithm that utilises the levels of knowledge provided by the requirements specification. Nevertheless, both of these methods for designing organisations are very complex and require an elaborate specification of the organisational requirements, while we just require a simple problem-solving organisation model. Moreover, they produce an instantiated organisation but not the generic model we need.

- **FORM:** In work that has similar aims to our own, Schillo et al. (2002) aim at an organisation framework that is flexible enough for self-organisation. However, they take a strictly emergent view of self-organisation and focus mainly on the social delegation aspects (gift exchange, voting and so on) in agent organisations. Furthermore, their method specifies a set of organisation models, and the participating agents choose, whether or not, to join such organisations. Therefore, this framework does not inherently aid the development of problem-solving agent organisations.

- **NMAS:** Another work that focuses on developing organisation models that permit reorganisation is by Vazquez and Lopez y Lopez (2007). They follow a norm based approach for modelling hierarchical agent organisations in which every role has a *position profile* associated with it. This profile is specified in terms of positional norms and an agent can take up a role by changing its own set of norms to conform to these positional norms. Therefore, their model allows the agents to change their organisational roles at run-time. However, the model requires that all positions and norms are specified at the outset itself. Moreover, such a model is useful for open systems with external agents, but is not required for the closed problem-solving organisations that we are currently interested in.

To sum up, the OperA, OMNI and Islander frameworks allow for an elaborate specification of an agent organisation and the interactions in it. The designer is expected to specify, in detail, the scenes and the interactions between the agents. These models are most suited for creating organisations in an open environment where external agents can enter, participate and leave the organisation. This is because they provide rigid guidelines for the organisation, which the participating agents need to obey, leading to regulation of the organisation. But they are not suitable for our purposes because the agents are not empowered to modify the organisation and have to abide by predefined structure and norms. They are also not apt for specifying organisation-level goals or activities as these organisations only restrict agent interactions, but do not strive to solve any problems or achieve any goals as such. On the other hand, the models that were developed to permit reorganisation (like the frameworks of Schillo et al. and Vazquez et al.) follow norm based approaches to enable the agents to change between specified roles. Therefore, any possible reorganisation process will be restrained to the few configurations visualised at design time. Moreover, these models are also not very suitable for problem-solving organisations in which the agents are internal to the system and with specified capabilities. The mathematical methodologies like ODML and KB-ORG specify frameworks to come up with efficient organisation designs given a quantifiable set of requirements. However, our set of requirements are too simple to warrant these complex algorithms. Moreover, they produce an instantiated, mathematically defined rigid organisation but not a generic model or framework of an organisation. Fortunately, the AGR and Moise organisation models are useful to develop goal driven organisations in which agents are part of the structure and perform some tasks. Furthermore, VDT aims to model a problem-solving agent organisation and hence the ideas used in VDT are appropriate for our first requirement, to model a simple problem-solving agent organisation which receives some task inputs, performs the tasks and returns the results. Therefore, the concepts introduced by AGR, Moise and VDT will form the primary basis for our organisation model.

2.1.3 Modelling Agents

An overview of modelling agents in the context of organisations is presented by Carley and Gasser (1999). From this, it is apparent that the modelling of agents varies across different organisation models. In particular, agents in the organisation may be homogeneous or belong to different classes. The agent's knowledge and cognition capability may be quite basic and primitive or highly sophisticated. The agents within the organisation may be selfless and cooperative or selfish and competitive. The abilities of the agents may be represented as a simple vector or as a complex combination of skills, decision strategies, preferences, modes of behaviour and so on.

Against this background, while all the organisation design approaches described in the previous subsection, with the exception of VDT, leave the agent development to the designer, VDT models the members of the organisation called actors in great detail. The main characteristics of the actors are attention allocation (determines the decision making behaviour of how the actor chooses among several task alternatives) and information processing (determines the skills, capacity and other processing characteristics). This design of agents will be partly used in our organisation model as it meets our requirements for modelling agents in the context of problem-solving organisations. Another concept that we will use is obtained from Gershenson (2007) where the agents are required to perform task assignment but can only address one request per time-step. Thus, we will also make use this of representation of agents possessing limited computational capacities so that efficiency of the agents plays a prominent role in the organisation performance.

2.1.4 Evaluating an Organisation's Effectiveness

Organisation characteristics play a major role in the performance of the organisation (Galbraith, 1977). Here, the criterion we will use for measuring the performance of the organisation is how well it performs its task (Fox, 1988) as we believe this provides a good indication of the organisation's efficiency.

Other evaluation methods include a qualitative comparison of the characteristics of commonly used organisation structures in multi-agent systems as presented by Horling and Lesser (2005). A somewhat quantitative method is presented by Grossi et al. (2006) for evaluating the structures of agent organisations. Using graph theory, they quantify the structural features of the organisation and then suggest how the values thus obtained can be used to analyse a number of properties of organisational structure (like robustness, efficiency and flexibility). However, both the above works are completely independent of the tasks that are being handled by the organisation. As a result, they fail to capture the suitability of a structure according to the task environment that it is situated in. Therefore, both these approaches are not appropriate for our second requirement which refers to developing an evaluation function for an organisation on the basis of the tasks executed by the organisation.

On the other hand, in VDT (see Section 2.1.1), the measure of the performance of the organisation is on the basis of the load on the organisation. The load on the organisation is represented in units of *work volume*, thereby providing a common calibration for different tasks. The total work volume of a task is taken as the sum of the production work volume and coordination work volume. The production work is divided into planned work (which is pre-defined in the task description) and production rework (arising due to exceptions). The production rework and the coordination work depend upon the characteristics of the organisation along with the particular task. Therefore, the resultant load on the organisation is a function of the tasks and the organisational characteristics and acts as an performance indicator. Thus, the evaluation method of VDT is compatible with our second requirement, as it is based on the task load of the organisation and, it will, therefore, be adopted in this work.

2.2 Self-Organisation in Multi-Agent Systems

As already noted, computing systems are becoming increasingly interconnected and more difficult to maintain (Horn, 2001). Due to the increase in the size, complexity and the number of components, it is no longer practical to anticipate and model all possible interactions and conditions that the system may experience at design time. Similarly, the systems are becoming too large and too complex for system managers to maintain them at run-time. To tackle these problems, a number of researchers have argued that such large complex systems should be autonomic — that is, the computing systems should manage themselves (Kephart and Chess, 2003; Mainsah, 2002). Specifically, autonomic systems are expected to maintain and adjust their operations according to changing requirements, demands, resources, other external conditions and failures. In short, autonomic systems possess the capability of self-management. The four aspects of self-management are *self-configuration, self-optimisation, self-healing* and *self-protection* (also called the self-* properties). Self-configuration in autonomic systems denotes that they should be able to autonomously configure and install or integrate themselves on the basis of some high level policies and prevailing system conditions. Self-optimisation means that the systems should continually try to be more efficient by monitoring and tuning some parameters, learning to make appropriate choices and adapting to their environment, thereby delivering a better performance. Autonomic systems are also expected to have self-healing properties and should be able to detect, diagnose and resolve any internal problems. Finally, the self-protection characteristic indicates that they should be capable of defending themselves against malicious attacks.

Moreover, this self-management behaviour of autonomic systems should arise in a decentralised manner, through the interactions between its individual components along with the internal self-management properties of the components. By so doing, the autonomic system will be more robust as there won't be a single point of failure. Such decentralised autonomic systems exhibiting the self-* properties, by definition, have to be autonomous and proactive. Therefore, a multi-agent systems approach is well suited for developing autonomic computing systems (Tesauro et al., 2004) as agents are also autonomous and proactive by nature[1]. Multi-agent systems also provide a suitable paradigm for decentralised systems in which autonomous individuals engage in flexible high-level interactions. Thus, the self-* principles of autonomic systems can be mapped onto the notion of agents by considering the components of the system to be autonomous agents engaging in interactions to produce an autonomous self-managed system (De Wolf and Holvoet, 2003). This naturally gives rise to the need for developing agents that are capable of displaying self-* properties both individually and collectively. Thus, it becomes necessary to explore multi-agent systems that can exhibit these self-management properties. The area of research that deals with this issue is self-organisation in multi-agent systems and, therefore, we will explore the different facets of this area in the remainder of this section.

2.2.1 The Basics

The concept of self-organisation is inspired from natural systems which function without any external control and adapt to changes in the environment through spontaneous reorganisation. This self-organising ability makes these natural systems robust to changing environmental conditions, thus enhancing their survivability. In the context of computing systems, self-organisation refers to the process of the system autonomously changing its internal organisation to handle changing requirements and environmental conditions (see Chapter 1 for the formal definition). It may involve creating an organisation from a set of unorganised agents or could mean reorganising an already existing organisation of agents or both, creating an organisation and continuously reorganising it as the environmental conditions vary. Further, self-organisation can be classified (Di Marzo Serugendo et al., 2005a) as:

- **Strong self-organising systems** — these function without any explicit central control
- **Weak self-organising systems** — these have a central controller or planner, internal to the system, that supervises the (re-)organisation process.

The concepts of strong self-organisation and emergence are closely coupled. Emergence is a phenomenon in which some properties and structure appear at the macro level which are not present at the micro level (Di Marzo Serugendo et al., 2005b, 2006). The structure appearing at the macro level is a result of the actions at the micro level, though no such order is observed at the micro level. Relevant examples include the

[1]Other approaches for autonomic computing include those based on reinforcement learning (Tesauro, 2007; Dowling et al., 2006). However, such RL mechanisms requires modelling the problems as Markov Decision Processes (MDPs) which may not always be possible because of incomplete observability and so on.

foraging action of ants and appearance of moving patterns in the game of life (Holland, 1998). Often, self-organisation, when occurring in a decentralised manner through local interactions, is an emergent phenomenon. However, it can exist without emergence and vice versa. For our purposes, we study self-organisation mechanisms, irrespective of whether they are an emergent phenomenon or not. Since strong self-organising systems do not have any central control and hence no single point of failure, they provide the most suitable paradigm for developing autonomic systems. In particular, they have the following characteristics:

- **No External Control:** The primary characteristic of self-organisation is that there is no external control of any kind. Reorganisation processes are initiated internally and result in changing the internal state of the system. Thus, the system manages itself. Also, the process of self-organisation implies that some kind of order be present in the resultant system after organisation or reorganisation.
- **Dynamic Operation:** A self-organising system is expected to evolve over time. Therefore, self-organisation is a continuous process. The property of self-organisation exists permanently in the system.
- **No Central Control:** Strong self-organising systems have no central authority to guide the reorganisation process. This lends robustness to the system as there is no single point of failure. Often, the organisation emerges through the local interactions of the individual components.

Thus, any strong self-organising approach will have to possess these characteristics. Next, we look at the different mechanisms in which self-organising systems can be developed.

2.2.2 Mechanisms of Self-Organisation

Several approaches have been explored by researchers for developing self-organising multi-agent systems. The different approaches can be classified on the basis of the mechanisms employed by them. Di Marzo Serugendo et al. (2006) identify the following categories of mechanisms:

1. **Direct interaction based mechanisms:** These are based on local interactions and computations of the agents that lead the system to converge to a coherent stable state. The focus is on generating an organisation from disordered agents. It is mainly applicable to the structural aspects of organisation like topological placement and communication of the agents (Mamei et al., 2004).

2. **Stigmergy-based mechanisms:** In a stigmergic process, global system behaviour emerges from the *indirect* interactions of the agents that occur by modifying the environment (Bourjot et al., 2003). It is difficult to predict the outcome of self-organisation methods based on these mechanisms as the global behaviour emerges through interactions with the environment.

3. **Reinforcement-based mechanisms:** In reinforcement mechanisms, a reward function catalyses the reorganisation. Agent behaviours are rewarded on the basis of some parameters and, consequently, agents adapt their behaviours to achieve better rewards. Therefore, self-organisation emerges from the adaptive behaviour of the agents. This mechanism is commonly used to create specialisations and divisions of labour among agents (Mano et al., 2006). Some reinforcement mechanisms, broadly classified as collective intelligence (COIN), follow a distributed reinforcement learning approach (Tumer and Wolpert, 2004). A *collective*, in this context, is a system in which the agents making up the system have private utilities, while the system has a global utility. Such a system is *factored* if increasing any agent's private utility cannot decrease the global utility. Furthermore, the system will have high *learnability* if any agent's actions do not affect the private utilities of the other agents. However, in general, a completely factored system cannot have complete learnability and vice-versa (Wolpert and Tumer, 2001). This is because if the system is perfectly factored and the agent's utility is precisely aligned with the global utility, then such a utility function of the agent will be highly influenced by the other agents that also influence the global utility thereby leading to low learnability. Therefore, achieving the requisite balance between the two is a major challenge.

4. **Cooperation-based mechanisms:** Self-organisation can be achieved through locally cooperative interactions between the agents that modify their behaviour on the basis of their local perceptions resulting in system-wide reorganisation. Locally cooperative interactions refer to agents behaving in such a way that they are benevolent towards other agents in the organisation. Two commonly used mechanisms are Organisational Self Design (OSD) and Adaptive Multi-agent Systems (AMAS). These two methods along with their advantages and drawbacks are discussed, in detail, in Section 2.2.3.

5. **Architecture-based mechanisms:** These mechanisms are based on the architectures or meta-models of the organisation. The commonly used method is *holarchies* which are hierarchies made up of holons. Holons are entities that can exist independently or can join with other holons to form bigger holons. Holons dynamically altering the holarchy according to changes in the environment, forms the basis of self-organisation (Bongaerts, 1998; Fischer, 2005). Therefore, holon based approaches focus on forming and disbanding groups of agents with a strict

hierarchy between the groups. Thus, while holarchies are always pyramidal in shape, we do not intend to place such constraints on the structures that might exist in an organisation.

These mechanisms have been used on several occasions, in different ways, to develop self-organisation systems. Since, we require reorganisation in explicitly defined problem-solving organisations, the agents will need to be cooperative and also have direct interactions with each other. Moreover, we intend to apply the mechanism of using past information for adaptation, somewhat similar to reinforcement methods. In the following subsection, we shall study a number of examples which have implemented self-organisation in multi-agent systems.

2.2.3 Examples of Self-Organisation in MAS

A number of applications have been developed that use self-organisation in multi-agent systems to address real-life problems like information retrieval, resource allocation and so on (Bernon et al., 2006). These applications employ a variety of techniques, ranging from those using reactive agents (actions directly triggered by percepts) and stigmergy, to those using agents that deliberate about cooperative situations. Self-organisation is displayed by virtual organisations (which do not have any organisation structure) and also by explicitly modelled organisations. While most self-organisation methods are emergent and decentralised, a few utilise a central controller internally. In the following, we shall study some of the implementations spanning across the various types of self-organisation mechanisms and discuss their usefulness with respect to our third requirement (stated in Section 1.1) which is developing a decentralised structural adaptation method for agent organisations. We divide the different implementations into those containing cooperative agents, self-interested agents, stigmergy, reinforcement and networks and, finally, those dealing with agent organisations (these do not particularly fit into any of the other classifications). We study each of them in turn.

First, we consider *self-organisation by cooperative agents*. One of the earliest multi-agent systems to employ self-organisation was developed by Gasser and Ishida (1991). It uses an organisation self design (OSD) mechanism to provide the agents with the ability to reorganise themselves. Specifically, OSD uses agent composition and decomposition to restructure the organisation. Gasser and Ishida employ OSD in a problem-solving organisation embedded in an environment. The agents split themselves into two or merge with a neighbour depending upon the changing conditions. The agents decide on the reorganisation actions (merging or splitting) on the basis of heuristic rules which are triggered by changes to the requirements or the environment (Ishida et al., 1992). This work was then extended by Kamboj and Decker (2006) to use a more detailed representation for tasks and resources. However, OSD mechanisms function by spawning and merging agents and would not be suitable in scenarios where the agents cannot be merged or divided. Hence, the OSD mechanism doesn't satisfy our requirements as we aim to develop reorganisation techniques that change the structure and norms of the organisation, but not the agents themselves.

Another self-organising mechanism for cooperative agents is based on Adaptive multi-agent systems (AMAS) theory (Picard and Gleizes, 2002). This theory aims to achieve self-organisation in problem-solving multi-agent systems through the locally cooperative actions of the agents (Capera et al., 2003a,b). The agents are supplied with skills, communication, knowledge about other agents and criteria to detect non-cooperative situations (NCS). The NCS are those that are adverse to the organisation. They are classified into three kinds— (i) incomprehensible signals from the environment, (ii) perceived information does not initiate any activity in the agent and (iii) the conclusions are not useful to others. The specific list of NCS needs to be pre-defined at design time by the designer. An agent on perceiving a NCS, tries to return to a cooperative situation through actions selected by its decision mechanism. Therefore, through the socially co-operative behaviour of the agents, an organisation emerges and is maintained. However, this approach relies on the designer being able to identify all possible non-cooperative situations and building the agents so that they handle them locally. Thus, this approach cannot be applied in environments where all the states of the organisation cannot be identified or classified at design time. Therefore, AMAS theory is also not suitable for our purpose.

Secondly, we move on to *self-organisation by self-interested agents*. While the above two approaches are based on agents that are cooperative in nature, self-organisation can also be present in multi-agent systems made up of self-interested agents. Virtual organisations (VOs) (Norman et al., 2004) are an example of such self-interested agents autonomously organising and reorganising into groups depending upon the circumstances in a market-place. Similarly, Knabe et al. (2003) use a holonic approach (see Section 2.2.2) to develop agents that form and disband virtual enterprises (VEs), according to their trade volume with the other agents. Both VEs and VOs are applicable in open dynamic environments wherein independent agents compete to provide services, but cannot be applied to a single problem-solving organisation of agents as they do not aim towards an efficient organisation as a whole. Self-organisation by competitive agents within an organisation is also used by Klein and Tichy (2006) to develop a fault-tolerant multi-agent system. In this case, fault tolerance is achieved through the agents dynamically reconfiguring their task specialisations to obtain better rewards. Therefore, every agent uses a simple decision theoretic approach by estimating the rewards for performing any of the other services and then chooses that service which predicts the highest reward to the agent. This reward function is designed such that when there are more agents than the demand requires, the reward is negative and vice versa. However, this work too is not suitable for our objectives as it is based on the assumptions that tasks do not have dependencies and that all agents have the ability to perform all services. While the ideas presented in this work are quite useful to us in terms of the agents reasoning based on rewards, they cannot be directly applied to our scenario as in our case, the primary goal of reorganisation is improving the efficiency of the organisation without changing the agents. Nevertheless, we will pick the idea of using a decision theoretic approach and use it in our method.

Thirdly, we consider *self-organisation by stigmergy, reinforcement and in networks.* Stigmergic and reinforcement mechanisms, mainly inspired from biology, have been used in reactive agents to develop self-organising multi-agent systems (Mano et al., 2006). The major problem with these mechanisms is that being emergent, the agent design does not guarantee particular global behaviour. Thus, the connection between local behaviours and global results is difficult to obtain. Therefore, the design of the agents is based on extensive experimentation to arrive at the correct parameters that result in useful global behaviour, thereby, making it an unreliable and lengthy approach.

In more detail, a stigmergic self-organisation approach that has been successfully applied in a multi-agent system is demonstrated by Schlegel and Kowalczyk (2007). They tackle the problem of resource allocation by proposing a distributed algorithm that does not require any central controller. Agents need to dynamically allocate tasks to servers that are shared between all the agents. The agents attempt to optimise their task allocations by forecasting the future task load on the servers on the basis of the history of server utilisation, obtained from the completed tasks at those servers. Every agent maintains a set of predictors per server. In every such set, one predictor is anointed as the *active predictor* and is used to forecast the future load on that server. On the basis of the forecasts on each of the servers, the agent chooses the server with the maximum capacity forecasted. Thus, the decision mechanism is based on standard decision theory. Also, using the feedback from the time taken to complete its tasks, the agent evaluates the *active predictor* for each server and switches to a different predictor, if necessary. The various strategies followed by the predictors are fixed at design time, only the method of selecting the active predictor is affected by the agent's history. In this way, efficient resource allocation emerges from the indirect interactions between the agents (as the agents only interact with the servers). Some of the ideas presented in this work, mainly the utilisation of the histories of task allocations and the use of decision theory, will be used in our reorganisation method. The major difference between this work and our requirements is that here, the agents do not interact directly and take all decisions independently; while in our model, the agents need to interact with each other to collectively decide about their relations. Furthermore, in this case, the self-organisation process influences the task allocations on a case-to-case basis, while we require self-organisation at the higher level of agent relations that, in turn, influence the task allocations.

Apart from stigmergic self-organisation seen in the biological domain, self-organisation that is seen in social and economic domains like trust behaviour of humans, gossipping and markets can also be applied to develop self-organising computing systems (Hassas et al., 2006). For example, the T-MAN protocol (Jelasity and Babaoglu, 2005) uses a gossip based mechanism to construct network topologies. The nodes are modelled as agents and select their neighbours through a ranking function that is based on local messages (gossip). This technique is useful to create an organisation, but not to dynamically adapt it according to changing conditions. Hence, it does not satisfy our criteria for a reorganisation method.

Network related problems provide a suitable scenario for employing self-organisation techniques. To this end, Mills (2007) presents a survey of various self-organisation techniques being used in wireless sensor networks. However, these methods are specific to network problems like query-routing and internal power management and cannot readily be ported to generic optimisation problems in multi-agent systems. Nevertheless, one particular work that is relevant to the current study is by Itao et al. (2002) in which autonomous components provide network services by forming relationships with other components based on a reward mechanism. The neighbour selection problem in self-organising networks can also be tackled by using a machine learning approach (Beverly and Afergan, 2007). Though the idea is generic, the implementation of the learning mechanism is specific to networks.

Finally, we deal with *self-organisation in agent organisations*. Though, the works described above can be applied to agent organisations, they do not deal with explicitly modelled organisations. In contrast, Horling et al. (2001) use a TÆMS representation to model an agent organisation and propose a diagnostic subsystem to be incorporated inside the agents. Such a subsystem would help the agents identify deficiencies in the organisation and suggest reorganisation measures. In particular, their proposed architecture has three layers — symptoms, diagnosis and reactions. Symptoms are observations of the environment, the diagnosis layer identifies deficiencies on basis of the symptoms, while the reactions layer suggests suitable measures based on the diagnosis. Though this work provides a means for the agents to detect the need for reorganisation, it does not elaborate on concrete reorganisation steps. A mapping between the diagnosis and the reactions is assumed. Thus, all the reorganisation steps have to be pre-designed which is not always possible. In a similar work, Hoogendoorn et al. (2007) present a formal description of the re-design process of organisations based on the AGR organisation model (see Section 2.1.2). Their work suggests an approach to represent the reorganisation process based on the requirements and goals of the organisation. However, it requires a global view of the organisation and does not explicitly specify how to reorganise either.

Dignum et al. (2004) discuss reorganisation in agent organisations by examining and classifying the various motivations for reorganisation and the different kinds of reorganisation possible. They broadly classify reorganisation into two types— (i) behaviour change involving short term behaviour modification of some agents and (ii) structural change involving long term changes in the structure of the organisation. Moreover, they emphasise on the necessity of concretely determining the complete utility of an organisation and its structure, which can thereby indicate the benefits of a given type of reorganisation. Thus, while their suggestions further justify our second requirement which refers to an evaluation mechanism for the organisations, they do not indicate any possible solutions.

A more detailed work on reorganisation that uses an explicit organisation model is by Hubner et al. (2004). Their work incorporates a controlled reorganisation mechanism into the MOISE framework (see Section 2.1.2). Controlled reorganisation follows a top-down approach in which a group of specialised agents carry out the reorganisation process which includes monitoring the organisation, designing the changes and implementing them. Thus, it is not bottom-up or completely decentralised as only some of the agents have reorganising capability. It also requires designing some agents with complex reorganisation modules. While this work addresses the same problem that we are interested in, the approach does not satisfy our requirements as we are interested in a completely decentralised approach in which all agents have reorganisation ability.

Bou et al. (2006a,b) also incorporate a reorganisation mechanism into the Islander organisation model (see Section 2.1.2). In it, a central authority named an 'autonomic electronic institution' modifies the norms of the organisation to accomplish institution level goals. Thus, this mechanism of self-organisation is centralised and is based purely on modifying the organisational level features like norms, without changing the agents or their relationships (structure). Another centralised mechanism developed by Hoogendoorn (2007) uses a max-flow network based approach to dynamically adapt organisations according to environmental fluctuations. It uses the AGR model (see Section 2.1.2) and specifies a mapping between this model and max-flow networks. In this case, the agents are regarded as nodes and their relationships as links of the network. The task requirements are modelled as the environmental pressure on the organisation which is mapped onto the source-sink paradigm of max-flow networks. The adaptation mechanism is based on identifying and adding capacities to the bottlenecks in the system and duplicating roles and the associated links to improve the max-flow of the system. However, again this approach requires a central authority to carry out the reorganisation.

Also, it aims at improving the capacity by adding links and nodes but does not attempt to optimise by removing redundant links or nodes.

Another such graph based approach for reorganisation is presented by Wang and Liang (2006). They represent the organisation structure using three graphs— (i) a role graph denoting the relations between the roles (ii) an agent graph, which is an instantiation of the role graph, depicting the relations between the agents depending on the roles allocated to them and (iii) a connector graph which links the agent graph to the role graph. The reorganisation process is based on graph transformation that occurs as agents shift between the roles. However, this transformation takes place according to predefined changes that correspond to different possible scenarios. Therefore, like AMAS, this method also requires that all the situations are anticipated at design time.

In recent work, Gershenson (2007) demonstrates a self-organisation approach for the problem of task assignment in agent networks. An agent, that receives a task, needs to send out some dependency requests to its neighbouring agents. Once it receives the responses to these dependencies, its task will be complete. In this way, every agent will receive several such dependency requests from its neighbours, which it stores in a queue and solves in a first-come, first-served basis. Furthermore, an agent can respond to only one request per time-step. Therefore, the performance of the network is measured by the number of tasks that are completed. This depends on the time 'wasted' by the agents waiting for the responses to their dependency requests from agents having long queues. The self-organisation process works by first identifying the agent (say A) with the longest queue. Then, among the agents dependent on A, the one with the largest waiting period chooses another agent (one with the shortest queue) to replace A as its neighbour. Therefore, the global knowledge of the queues of every agent is required in this method. This does not conform with our requirements, as in our case, the agents will only possess local information. However, another method that is mentioned in this work, but not expanded, relates to dynamically creating direct links between frequently interacting nodes to 'cut out' the intermediaries and shorten the communication time. This idea, originally presented by Bollen and Heylighen (1996) in the context of the world wide web, will be built up on in our adaptation process.

After analysing the various self-organisation techniques in multi-agent systems, we find that most of them cannot be applied to explicitly modelled problem-solving organisation of agents. This is because they cannot be incorporated in deliberative agents working towards common goals as are present in such organisations. Those that can be applied either self-organise by creating and deleting the agents (OSD) or by enumerating all the possible scenarios (AMAS). But, our third requirement is to develop decentralised reorganisation techniques without the addition/deletion of agents in a non-deterministic environment. Furthermore, the few self-organisation approaches that have been applied on organisation models of agents are centralised in nature, while we seek a completely decentralised approach in which all the agents are capable of reorganisation without holding the complete view of the organisation. Nevertheless, we can still reuse the ideas of agents being cooperative to each other, decision theoretic approaches for the reasoning of the agents and changes in agent behaviour based on history of allocations to develop our reorganisation method.

2.3 Summary

In this chapter, we have studied several existing approaches for modelling agent organisations which include modelling the tasks, the agents and the organisation characteristics (like structure and norms). We then examined some methods for evaluating the performance of an organisation. Next, we provided the definition and features of self-organisation in multi-agent systems. We followed that by enumerating the various mechanisms of self-organisation. Finally, we concluded by providing a survey of the various techniques of self-organisation that have been developed and used in multi-agent systems and analysed them with respect to our requirements.

To sum up, we find the PCANS model for task representation suitable for our needs and similarly, the agent model of VDT is appropriate for designing the agents. However, for organisation modelling, we will pick up ideas from a number of organisation models like AGR, MOISE and VDT because none of them individually satisfy all our requirements. On the other hand, most of the self-organisation techniques that we studied do not meet our requirements, as either they cannot be applied to agent organisations (like stigmergy or network related) or they work by modifying the agents. Those that reorganise by changing the structure or norms of agent organisations are centralised in nature. But, as already stated, we want to explore completely decentralised methods for reorganisation. Nevertheless, we can still reuse some of the concepts of self-organisation in cooperative agents and reorganisation mechanisms based on rewards or history to obtain the same kind of structural reorganisation as displayed by the centralised approaches in agent organisations.

We now move onto our model of agent organisation and reorganisation techniques. The next chapter presents our organisation model in detail. The following chapter introduces our reorganisation method along with the accompanying assumptions and constraints. The remaining chapters after that discuss the experimental results and conclusions.

Chapter 3

Modelling Agent Organisations

This chapter describes our model of an agent organisation. As mentioned in Section 2.1, this involves modelling the task environment, the agents and the organisational characteristics. Furthermore, it also presents a method for evaluating the performance of the organisation in terms of its efficiency. Therefore, this chapter addresses the first two requirements detailed in Section 1.1 which are developing a framework for problem-solving agent organisations and methods for evaluating such an organisation's performance.

In more detail, the purpose of our organisation model is to serve as a platform for demonstrating reorganisation techniques (see Section 1.1). Thus, for now, we develop a minimal organisation model without attempting to include all the possible features that an organisation might have. Specifically, our model of an agent organisation is a problem-solving group of agents situated in a task environment. By problem-solving agents, we mean agents that receive some input (task), perform some actions on the basis of that input (processing or execution) and return a result. Correspondingly, the task environment presents a continuous dynamic stream of tasks to be performed. This environment also has other parameters, independent of the task stream, which have a bearing on the organisation. These can be considered to be the costs associated with the environment. The task stream and the environmental costs are used as the basis for evaluating the performance of the organisation. So, we proceed by first describing the task environment, then the agent organisation and finally the evaluation mechanism.

Specifically, the next section details our task representation. The following section presents our model of the organisation and its characteristics by describing the agents and the organisation structure. Our method for evaluating the efficiency of the organisation for a given set of tasks and environmental parameters is explained in the third section. The final section summarises the chapter. A glossary of the terms introduced in this chapter is given in Appendix A.

3.1 Task Representation

The task environment contains a continuous stream of tasks that are to be executed by the organisation. A task can be presented to the organisation at any point of time and the processing of the task must start immediately from that time-step. Thus, the organisation of agents is presented with an incoming stream of tasks that they should accomplish. In detail, the organisation of agents provides a set of services which is denoted by S. Every task requires a subset of this set of services. Services are the skills or specialised actions that the agents are capable of. We model tasks as a composition of several service instances in a precedence order. We define a service instance si_i to be a 3-tuple: $\langle s_i, p_i, n_i \rangle$ where $s_i \in S$ (i.e. s_i is a member of the services set S), $p_i \in \mathbb{N}$ denotes the amount of computation required per time-step (computational rate) of the particular service s_i and $n_i \in \mathbb{N}$ denotes the number of time-steps that the service s_i should be provided to accomplish this service instance. Therefore, an agent taking up this service instance *has* to execute it at the specified rate p_i for the specified time-steps n_i (as opposed to, say, executing it at half the rate and double the time-steps). SI denotes the set of all service instances.

Following the PCANS model of task representation (see section 2.1.1), we only consider sequential dependencies between the service instances. Thus, the service instances of a task need to be executed following a precedence order which is also specified in the task representation. We model the precedence order as a tree structure in which the service instances that form the child nodes of a particular service instance need to be executed first before that service instance (parent node) can be executed. Thus the parent node is sequentially dependent on its child nodes. This goes on recursively until there are service instances which have no other child nodes (no sequential dependencies). These will be the leaf nodes of the tree structure. Thus the root node or the final service instance will have to be executed last in a task.

Thus, a task t_i is defined as a tuple containing a set of service instances and a set of dependency links:

$$t_i = \langle \{si_j \in SI\}, D_i \rangle \tag{3.1}$$

where D_i is the set of dependency links containing links between the various si_j of the task. These links are directed arcs between any two service instances depicting a sequential dependency of the source on the destination. So an element d_j of D_i is of the form: $d_j = \langle si_x, si_y \rangle$ where si_x and si_y are the service instances at the origin and the destination of the link.

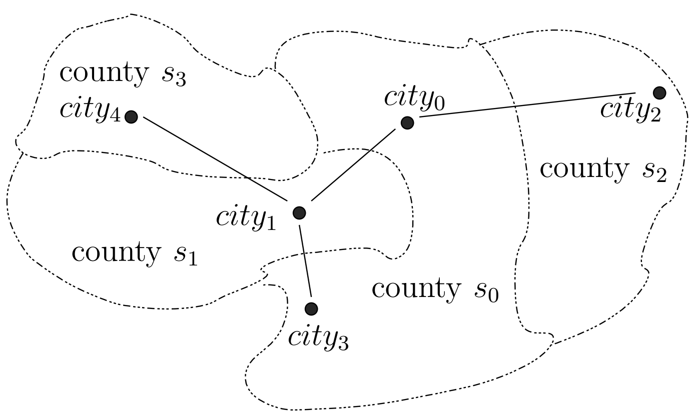


FIGURE 3.1: Map of the cities, their counties and the possible route

3.1.1 Example

To illustrate our task model, we will use it to represent a task derived from the domain of distributed case-based reasoning (Plaza and McGinty, 2005). Collaborative case-based reasoning is a form of distributed case-based reasoning, in which cooperative agents with different problem-solving experiences jointly find solutions to given problems. McGinty and Smyth (2002) use such an approach to tackle the problem of personalised route planning. In more detail, let us assume that a user needs a route starting from $city_0$ and covering $city_1$, $city_2$, $city_3$ and $city_4$. Also, we assume that finding a route to a city requires geographical knowledge about the county containing the city. This knowledge about the counties can be considered to be the services provided by the agents. Now, supposing that $city_0$ and $city_3$ belong to the same county, they will require the same service (say s_0), while the others belong to different counties, thereby, requiring different services (see Fig 3.1).

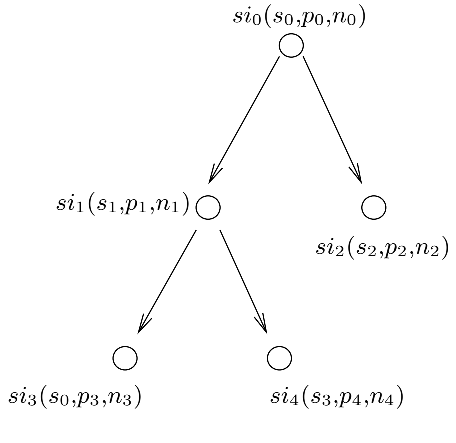


FIGURE 3.2: Representation of an example task

In terms of our model, this will translate to five service instances (si_0, si_1, si_2, si_3 and si_4) each representing a city ($city_0$, $city_1$, and so on) and requiring services s_0,

s_1, s_2,s_0 and s_3 respectively. As a case-based reasoning approach is used to find routes personalised according to the users' preferences, the process of obtaining a route to a city within a given county will require some amount of computation and some period of time (for possible iterations of selection). Assuming that these details are also provided, we can also supply our service instances with the specified computational rate (denoted by p_0, p_1, and so on) and the specified time duration (t_0, t_1, and so on) that the computation is required. Furthermore, finding a route to a city within a county also depends on the entry and exit points to the county from the neighbouring counties. For instance, finding the route within a county is dependent on the route from the border of this county to the destination city in the neighbouring county. Therefore, our service instances will also have dependencies within them. The task structure including the service instances and the dependency links is shown in Fig 3.2 (an arrow indicates that the service instance at the source node can only be executed once the service instance at the end node is completed). Since the route requires going to $city_2$ from $city_0$, the internal route in the county of $city_2$ determines how the route from $city_0$ should approach the common border; si_0 is dependent on si_2. Similarly, si_0 is also dependent on si_1, which in turn, is dependent on si_2 and si_3.

Therefore, the dependency links set D_0 contains four elements representing the four arrows shown. Thus:

$$D_0 = \{\langle si_0, si_1\rangle, \langle si_0, si_2\rangle\ \langle si_1, si_3\rangle, \langle si_1, si_4\rangle\}$$

Thus, the task tuple is:

$$t_0 = \langle\{si_0, si_1, si_2, si_3, si_4\}, D_0\rangle$$

In summary, our model of the task environment consists of a stream of tasks in which each task is made up of a set of service instances and a set of dependency links between the service instances. Every service instance specifies the service, the computational rate and the number of time-steps that computation is required. Next, we describe the representation of our agent organisation.

3.2 Organisation Representation

Since, we aim to model a problem-solving agent organisation, our organisation model consists of a set of cooperative agents. An agent is an independent computational entity that can provide one or more services. We model our agents by simplifying the agent model used by VDT (see Section 2.1.3) and consider only the information processing characteristics of the agents by overlooking the attention allocation characteristic. The attention allocation characteristic enables an agent to schedule its allocated tasks. However, this aspect is internal to an agent and completely independent of the organisation structure which is our primary focus. Therefore, we simply assume that the agents do not have a choice and have to execute all the service instances allocated to them on a first-come-first-served basis.

In more detail, the agents are associated with particular sets of services. These sets can be overlapping, that is two or more agents may provide the same service. Also, building on the agent model used by Gershenson (see Section 2.1.3), every agent also has a computational capacity associated with it. The computational load on an agent (explained later), in a time-step, cannot exceed this capacity. Formally, let A be the set of agents in the organisation. Every element in this set is a tuple of the form:

$$a_x = \langle S_x, L_x \rangle \tag{3.2}$$

where the first field, $S_x \in S$ denotes a set of services that belong to the complete service set S and $L_x \in \mathbb{N}$ denotes the capacity. The agents, their service sets and their capacities may change during the lifetime of the organisation.

The other features of an agent organisation, in general, are its structure and norms (see Section 3.2.1). The structure of an organisation represents the relationships between the agents in the organisation, while the norms govern the kind of interactions and messages possible between the agents. However, since we are developing a problem-solving organisation, the agents are all internal to the organisation and share the same goals. Moreover, all the agents will be designed in the same way, and therefore, their interaction protocol will be similar and can be internally specified. Therefore, an explicit definition of norms is not required to regulate their interactions. Thus, in our model, the relationships between the agents (denoted by the structure) also determine the interactions between the agents. Formally, an organisation is defined as consisting of a set of agents and a set of organisational links. It can be represented by a 2-tuple of the form:

$$ORG = \langle A, G \rangle \tag{3.3}$$

where A, as stated above, is the set of agents, G is the set of directed links between the agents (will be described later in this section).

As mentioned in the previous section (3.1), the organisation is presented with a continuous stream of tasks which are completed by the agents through their services. Tasks come in at random time-steps and the processing of a task starts as soon as it enters the system. Task processing begins with the assignment of the final service instance (root node) to a randomly selected agent. An agent that is ready to execute a particular service instance is also responsible for the allocation of the service instances on which it is dependent (as specified by the dependency links of the task) to agents capable of those services. Thus, the agents have to perform two kinds of actions: (i) allocation of service instances and (ii) execution of service instances. For instance, if the service instance that an agent is ready to execute is dependent on two other service instances, then that agent needs to find and allocate appropriate agents to execute those two service instances, obtain the results of the executions, then execute this service instance and finally send back its result. Moreover, every action has a load associated with it. The load incurred for the execution of a service instance is equal to the computational rate specified in its description (the number of computational units it consumes in a time-step), while the load due to allocation (management load) depends on the relations of that agent (will be explained later). As every agent has a limited computational capacity, an agent will perform the required actions in a single time-step as long as the cumulative load on the agent is less than its capacity. If the load exceeds the capacity and there are actions still to be performed, these remaining actions will be deferred to the next time-step and so on. Also, the service instances received by an agent are performed (allocated or executed) on a first-come first-served basis (no kind of scheduling is permitted). In contrast, there is no limit to the number of messages (to communicate with its relations (explained later)) that agents can send or receive in a single time-step. However, the messages received in a particular time-step can only be interpreted by the agent in the next time-step.

Next, we present, in detail, our representation of the organisation structure.

3.2.1 Organisation Structure

As stated earlier, agents need to interact with one another for the allocation of service instances. The interactions between the agents are regulated by the structure of the organisation. We base our model of organisation structure on the one described in Moise (see Section 2.1.2). Specifically, we adopt its organisational links paradigm, but ignore the missions associated with the links because, otherwise, the links will be task-specific. This is not possible in our model since the agents will not know the whole set of tasks at the beginning itself. We assume that the organisational links are valid irrespective of the tasks being executed by the agents. Therefore, the structure of our organisation can be described using three graphs — (i) *acquaintance graph*, (ii) *control graph* and (iii) *communication graph*. The nodes in each of these graphs represent the agents of the organisation. We now describe each one of the graphs in turn.

Acquaintance Graph: A directed graph in which an agent has a directed edge to all other agents whose presence is known to it. The graph represents the acquaintance relations between the agents of the organisation. Thus the acquaintance graph of an organisation has to be a connected graph, as there should exist a path between any two agents. Thus, an agent has a directed edge to each of its *acquaintances*. The agent is unaware of the presence of the other agents with whom it has no edge. They are called *strangers* with respect to this agent.

Control Graph: A directed graph which has two types of edges denoting two kinds of relations — authority and peer. A directed edge originates from one agent to another if the first agent is a *superior* of the second. Therefore, the control graph depicts the authority relations in the organisation. These superior-subordinate links can also be called authority links. The other kind of links that will be present in the control graph are called peer links. These will be edges between agents that are considered to be *peers* of each other. The peer relationship will be present among agents who are considered equal in authority with respect to each other and is useful to occasionally cut across the hierarchy. Generally, it is expected that an agent will interact more frequently with its subordinates and superiors than its peers.

The control graph is a sub graph of the acquaintance graph. Thus, every edge present in the control graph must also be present in the acquaintance graph, though the converse needn't be true (every pair of acquaintances needn't necessarily have an authority of peer relationship). That is, for an agent to control or be in the control of another agent, it must be aware of the presence of that agent. Similarly, an agent cannot consider another agent to be its peer without being acquainted with it. We term two agents having only an acquaintance relation (and not having an authority or a peer link) between them as *pure acquaintances* of each other.

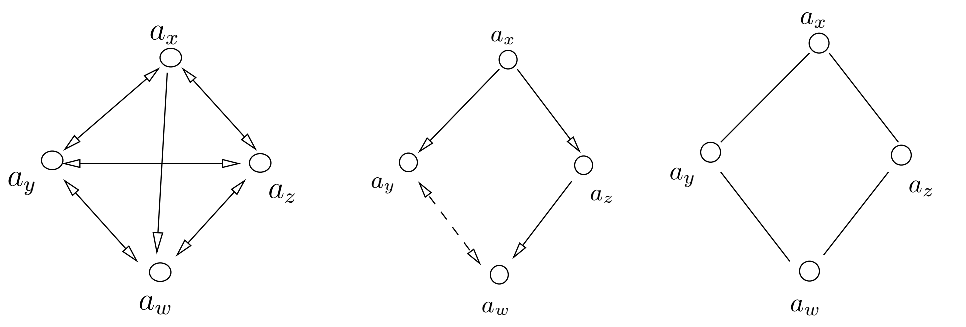
Following this, we can classify the various relationships that can exist between agents into four types — (i) stranger (not knowing the presence), (ii) pure acquaintance (knowing the presence, but no interaction), (iii) peer (low frequency of interaction) and (iv) superior-subordinate (high frequency of interaction). The type of relation present between two agents determines the information that they hold about each other and the interactions possible between them[1]. The information that an agent holds about its relations is:

1. The set of services provided by each of its peers (S_y of each peer a_y)
2. The accumulated set of services provided by each of its subordinates. The *accumulated service set* of an agent is the union of its own service set and the accumulated service sets of its subordinates, recursively. Thus, the agent is aware of the services that can be obtained from the sub-graph of agents rooted at its subordinates though it might not know exactly which agent is capable of the service. We denote the accumulated service set of an agent a_x as AS_x.

[1]For our purposes, we ignore the social obligation aspects of the relations as we are dealing with cooperative agents and also because we assume that an agent has to process all the services instances received by it irrespective of the sender being a superior or a peer

Furthermore, the mechanism for allocating service instances to other agents is also based on the agents' relations (explained later in this section). Note that, to avoid an infinite loop of task delegation (due to circular loop of authority), the authority relations are not permitted to have cycles. In summary, the authority relations impose the hierarchical structure in the organisation, while the peer relations enable the otherwise disconnected agents to interact directly.

Communication Graph: An undirected graph that represents the existing communication links between the agents. An agent can send a message to only those agents that are directly linked to it in the communication graph. It is also a sub graph of the Acquaintance graph. Thus, superior-subordinate agents and peer agents will have communication links between them.



(a) Acquaintance Graph (b) Control Graph (c) Communication Graph

Figure 3.3: An example organisation structure

All the graphs described above can be combined to form a single Organisation Graph, denoted by G, containing a set of organisational links (relations). Every link r_i that belongs to G is of the form:

$$r_i = \langle a_x, a_y, type_i \rangle \tag{3.4}$$

where a_x and a_y are agents that the link originates and terminates at respectively and $type_i$ denotes the type of link and can take any of the values in the set $\{Acqt, Supr, Peer\}$ to denote the type of relation existing between the two agents (the value $Acqt$ denotes a pure acquaintance relationship). The absence of a link between two agents means that they are strangers (no relation).

To illustrate the framework, consider a sample problem-solving agent organisation that is performing the personalised route planning tasks, described in Section 3.1.1, using a case-based reasoning approach, as stated earlier. Taking a limited view, let this organisation have four agents — a_x, a_y, a_z and a_w. The services provided by an agent are basically the knowledge bases contained by it about the routes in the various counties. Therefore, let us assume that a_x contains information about county s_0 and hence provides service s_0. Similarly, a_y provides service s_1, a_w provides s_2 and a_z provides s_3. Moreover, since

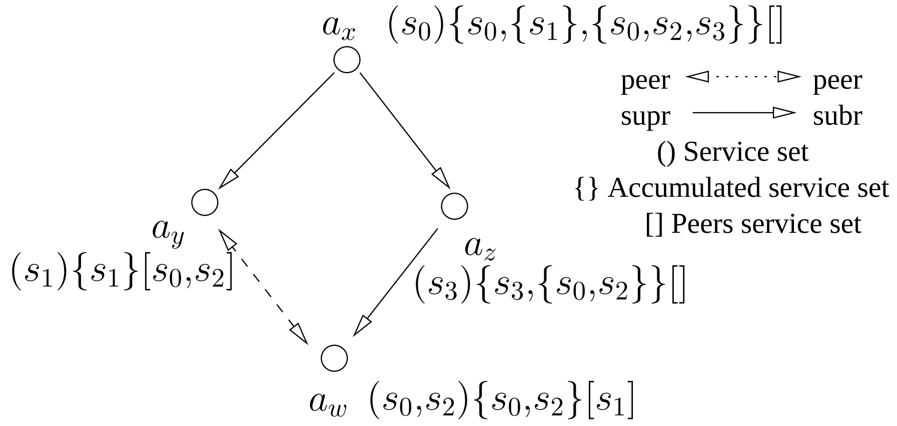

a_x $(s_0)\{s_0,\{s_1\},\{s_0,s_2,s_3\}\}[]$
a_y $(s_1)\{s_1\}[s_0,s_2]$
a_z $(s_3)\{s_3,\{s_0,s_2\}\}[]$
a_w $(s_0,s_2)\{s_0,s_2\}[s_1]$
peer peer
supr subr
() Service set
{} Accumulated service set
[] Peers service set

FIGURE 3.4: An example organisation graph

s_0 is a big and densely populated county generating a lot of queries with a long border with s_2, a_w contains information about s_0 in addition to s_2.

Given this, let us look at the possible structure of the organisation. Let a_y and a_w have a peer relationship. Also, assume a_x has two subordinates — a_y and a_z (because a lot of users need routes from cities in s_0 to s_1 and s_3). a_z, in turn, has a_w as a subordinate. Moreover, all the agents have an acquaintance relationship with one another, except a_w which is not aware of a_x. For this organisation, the three graphs are shown in Fig 3.3. In the control graph, the continuous arrow indicates a superior-subordinate relationship (superior at the start of the arrow) and the dashed arrow represents a peer relationship. The G for this organisation contains 8 organisational links which are 3 superior links, 2 peer links and 3 acquaintance links. Thus:

$$\begin{aligned} G = \{&\langle a_x, a_y, Supr\rangle, \langle a_x, a_z, Supr\rangle, \langle a_z, a_w, Supr\rangle, \langle a_y, a_w, Peer\rangle, \\ &\langle a_w, a_y, Peer\rangle, \langle a_y, a_z, Acqt\rangle, \langle a_z, a_y, Acqt\rangle, \langle a_x, a_w, Acqt\rangle\} \end{aligned}$$

In addition, the information possessed by the agents about the services provided by their relations is shown in Fig 3.4. The parenthesis denote the services provided by the agent itself, the curly braces contain the accumulated service set of the agent and the square brackets contain the services provided by the peers. For example, the accumulated service set of agent a_x, in turn, contains three sets representing its own service (s_0), the accumulated service set of its subordinate a_y (s_1) and the accumulated service set of its other subordinate a_z (s_0, s_2, s_3).

Against this background, we will explain the process followed by an agent for allocating service instances to other agents and how this mechanism is primarily influenced by the organisation structure.

3.2.2 Agent Decision Mechanism

As discussed earlier, agents need to perform the allocations of service instances along with the executions. During task execution, whenever any agent requires a service instance to be executed (either because it has been assigned that service instance or because the service instance forms a dependency of another service instance that is being executed by the agent), it first checks whether it is personally capable of the service and whether it has available computational capacity (not completely filled by the load) to perform the service instance. If so, it delegates the service instance to itself and proceeds towards executing it. However, if the agent does not possess that service capability or has no available capacity, it iterates over all its subordinates to find the prospective agents to whom the service instance can be assigned. Once it obtains the set of suitable subordinates, it randomly assigns the service instance to one of them (since the agents do not possess information about the available capacities of the other agents, they do not seek optimal distribution of the load and just allocate randomly to suitable agents). In this case, a subordinate qualifies for being assigned the service instance if it contains the required service in its accumulated services set. If there are no suitable subordinates (no subordinate or their subordinates are capable of the service) and it is capable of the service itself (but did not initially assign to self because its capacity is filled), then it will add the service instance to its waiting queue for execution. However, if it is not capable of the service (and nor are its subordinates), the agent proceeds, as above, by iterating over its peers to find the set of prospective agents and randomly assigning it to one of them. However, in this case, it only checks the set of services directly provided by the peers (since it does not hold information about the accumulated services sets of the peers). If it doesn't find any suitable peers either, it randomly assigns the service instance to one of its superiors (considered as *handing back* because it could not find a suitable agent). On the occasions when it does not have any superiors, it checks among its acquaintances for a suitable agent and tries to form a subordinate relation with it, if that doesn't result in a cycle of authority links (a cycle may result in an unending loop of assignment and hence should be avoided). Otherwise, it forms a peer relation with that acquaintance. One unit of management load is added to the load on the agent every time it considers an agent for an assignment (explained in detail in Section 3.3). It should be noted that every service instance will eventually find an agent that will execute it and therefore all tasks are completed by the organisation.

An agent that is ready to execute a particular service instance proceeds by first checking whether it has any dependencies. If it does, the agent assigns each of the service instances forming the dependencies to other agents (this may include itself) following the procedure detailed above. When assigning a service instance, it first sends an assignment message with the service instance details to the assigned agent. Every assigned agent

```
input: Service instance $si_j = \langle s_j, p_j, n_j \rangle$, Agent $a_x = \langle S_x, L_x \rangle$
if $s_j \in S_x$ AND $L_x - l_x \geq p_j$ then            // $l_x$ is the load on the agent
    Process($si_j$);                                        // delegation to self
end
listOfSuitableAgents ← ∅;
foreach subordinate $a_y$ do
    if $s_j \in AS_y$ then        // $AS_y$ is the accumulated service set of $a_y$
        listOfSuitableAgents.Add($a_y$);
    end
end
if listOfSuitableAgents == ∅ AND $s_j \in S_x$ then
    Process($si_j$);        // delegation to self, despite filled capacity
    Return();
end
if listOfSuitableAgents == ∅ then
    foreach peer $a_y$ do
        if $s_j \in S_y$ then                    // $S_y$ is the service set of $a_y$
            listOfSuitableAgents.Add($a_y$);
        end
    end
end
if listOfSuitableAgents == ∅ then
    foreach superior $a_y$ do
        listOfSuitableAgents.Add($a_y$);
    end
end
if listOfSuitableAgents ≠ ∅ then
    assignedAgent ← randomly selected from listOfSuitableAgents ;
end
foreach acquaintance $a_y$ do                // finding suitable acquaintances
    if $s_j \in S_y$ then
        listOfSuitableAgents.Add($a_y$);
    end
end
assignedAgent ← randomly selected from listOfSuitableAgents ;
Form_relation_with(assignedAgent);
Assign($si_j$,assignedAgent);        // 'Assigned' function of assignedAgent
```

Algorithm 1: Assigned (si_j): assignment of a service instance si_j by agent a_x

```
input: Service instance $si_j = \langle s_j, p_j, n_j \rangle$
dependenciesList ← Get_dependencies_of($si_j$);
foreach $si_k \in$ dependenciesList do
    Assign($si_k$,self);            // start by assigning dependencies to self
end
obtainedResultsofAllDependencies ← FALSE;
while obtainedResultsofAllDependencies $\neq$ TRUE do // waiting for results of
    obtainedResultsofAllDependencies ← TRUE;   // execution of dependencies
    foreach $si_k \in$ dependenciesList do
        if Waiting_for_result($si_k$) then
            obtainedResultsofAllDependencies ← FALSE;
        end
    end
end
if $L_x - l_x \geq p_j$ then
    $T_{x_E}$.Enque($si_j$);                  // adds to the tasks execution list
else
    $T_{x_W}$.Enque($si_j$);                    // adds to the tasks waiting list
end
```

Algorithm 2: Process (si_j): processing of a service instance si_j by agent a_x prior to its execution

sends back an acknowledgement message comprising a list of details of the agents forming the assigned agents after it, ending with the last assigned agent (which is always the agent that is actually executing the service). The agent executing the service instance (also called the delegated agent for that service instance) sends back the result of the execution to the agent that assigned the service instance to it and this result is similarly passed back, in sequence, by all the assigned agents to finally reach the agent that first required the execution of the service instance. An agent can start the execution of its delegated service instance only after it obtains the results of all the service instances that form the dependencies of the particular service instance. Every message (assignment, acknowledgement and result) has a communication cost associated with it (explained in Section 3.3) The decision mechanism described above is presented in the form of a pseudocode in Algorithms 1 and 2.

Considering the sample organisation in Fig 3.4 executing the task in Fig 3.2, the distribution of service instances across the agents occurs as shown in Fig 3.5. In detail, we assume that the task arrives at agent a_x (randomly selected). Hence, a_x checks that it is capable of the final service instance si_0 (as it is capable of service s_0 and has available capacity) and therefore, allocates si_0 to itself. In the next time-step, a_x needs to allocate the two dependencies of si_0 which are si_1 and si_2 to capable agents. For allocating si_1, it checks the accumulated service sets of its two subordinates (a_y and a_z) and allocates to a_y (because it is the only one capable of service s_1). Similarly, it allocates si_2 to a_z because this subordinate contains service s_2 in its accumulated service set. In the same

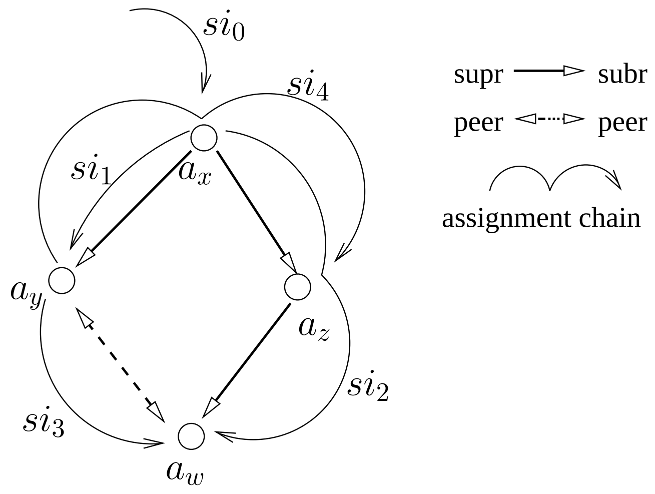


Figure 3.5: Distribution of service instances of the task in Fig 3.2 across the agents

way, before a_y can execute si_1, it needs to obtain the results of the two dependencies (si_3 and si_4) by allocating them to appropriate agents. At the same time, a_z has to reallocate si_2 to its subordinate because it is not capable of the service s_2 itself. So, a_y allocates si_3 to its peer a_w as it has no subordinates. It also hands back si_4 to its superior a_x as it has found no suitable subordinates or peers for that service. Meanwhile, a_z allocates si_2 to its subordinate a_w which then proceeds to execute it. Moreover, a_x assigns the service instance si_4 requested by a_y to its subordinate a_z (capable of service s_3) which then proceeds to execute it.

Thus, the structure of the organisation influences the allocation of service instances among the agents. To sum up, we present our organisational model by representing the agents, including their allocation mechanism, and the organisation structure. In the next section, we study our method for evaluating the performance of the organisation based on its efficiency.

3.3 Organisation Performance Evaluation

Before designing any kind of adaptation techniques, there needs to be a mechanism that can evaluate the performance of an organisation for some given set of tasks (see Section 1.1). Towards this end, we introduce the concept of load, cost and benefit of an agent and thus also of the organisation in total. The total cost and benefit of the organisation are considered to be the two parameters measuring the efficiency of the organisation. Decreasing the cost or increasing the benefit will mean improving the efficiency and vice versa. These load, cost and benefit values are calculated for every time-step throughout the lifetime of the organisation and hence provide a good measure of the performance of the organisation.

In greater detail, our evaluation mechanism is essentially the one used by VDT (see section 2.1.4) and is of two types — task-related (production work in VDT) and management-related (coordination work in VDT). We further simplify it by not considering any production rework and denoting coordination work as just the allocation of the dependencies in a task.

The cost of the organisation is based on the amount of resources being consumed by the agents. In this case, we consider the cost of an agent to be equal to the network resources utilised to transmit its messages. Therefore, cost to the organisation due an agent a_x, for a time-step, is:

$$cost_x = C.c_x \tag{3.5}$$

where c_x is the number of messages sent by that agent in that time-step and C is the communication cost coefficient. C represents the proportion of a computational unit required for the transmission of a single message by the network. Moreover, the cost of the organisation is the sum of the costs of all the messages being transmitted in that time-step (obtained by adding the individual agents costs from Eqn 3.5):

$$cost_{ORG} = C.\sum_{x=0}^{|A|} c_x \tag{3.6}$$

where A is the set of agents in the organisation.

Unlike the cost, which is dependent on the number of messages, the benefit of the organisation is affected by the load on the agents. As stated earlier, agents have limited capacities and their computational load cannot increase beyond this capacity. The load on a_x, in a time-step, is calculated as:

$$l_x = \sum_{i=0}^{|T_{x_E}|} e_{i,x} + M \sum_{i=0}^{|T_{x_F}|} m_{i,x} \tag{3.7}$$

- $e_{i,x}$ is the amount of execution computation of a_x required by task t_i
- $m_{i,x}$ is the amount of management computation done by a_x for task t_i
- T_{x_E} is the set of tasks being executed by a_x
- T_{x_F} is the set of tasks being assigned by a_x
- M is the management load coefficient

More specifically, $e_{i,x}$ represents the sum of the computational rates of all the service instances of task t_i that are being executed by a_x in that time-step. Similarly, $m_{i,x}$ is the amount of management computation done by a_x for deciding the allocation of the required service instances of task t_i to other agents. It is equal to the sum of the number of agents (relations) that a_x had to consider for the allocation of the service instances of t_i in that time-step. Also, the management load coefficient M represents the proportion of a computational unit used up by the agent when it evaluates whether an agent can be assigned a service instance.

This load l_x on a_x cannot exceed its capacity L_x ($l_x \leq L_x$). Any excess tasks will be waiting for their turn in the sets T_{x_W} and will adversely affect the benefit of the agent. Therefore, the benefit value reflects the speed of completion of the tasks. Hence, the benefit of an agent, at a time-step, is equal to the total amount of computation of the service instances being executed by that agent subtracted by the amount of computation required by the service instances that are waiting to be executed or allocated by the agent. Therefore,

$$\mathit{benefit}_x = \sum_{i=0}^{|T_{x_E}|} e_{ix} - \sum_{i=0}^{|T_{x_W}|} e_{ix} \tag{3.8}$$

In more detail, T_{x_W} represents the set of tasks that have service instances waiting to be either executed by a_x or allocated by a_x to appropriate agents. Therefore, to obtain the maximum possible benefit, the agent should never have any waiting tasks (it should never be overloaded) and the assignment of the dependent service instances should occur in the same time-step that they are discovered as dependencies. Furthermore, similar to the organisation cost, the total benefit of the organisation is simply the sum of the individual agent benefits (from Eqn. 3.8):

$$\mathit{benefit}_{ORG} = \sum_{x=0}^{|A|} \mathit{benefit}_x \tag{3.9}$$

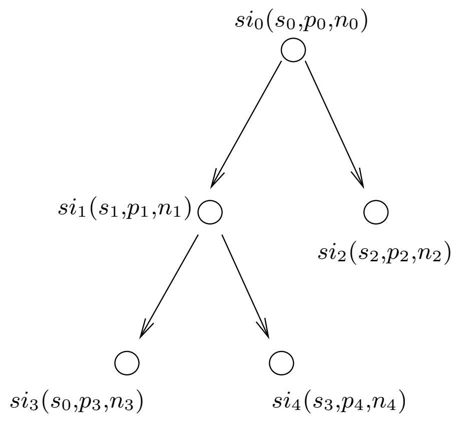
Since, the benefit should be maximised while the cost needs to be minimised, the overall efficiency of the organisation is measured as

$$\mathit{efficiency}_{ORG} = \mathit{benefit}_{ORG} - \mathit{cost}_{ORG} \tag{3.10}$$

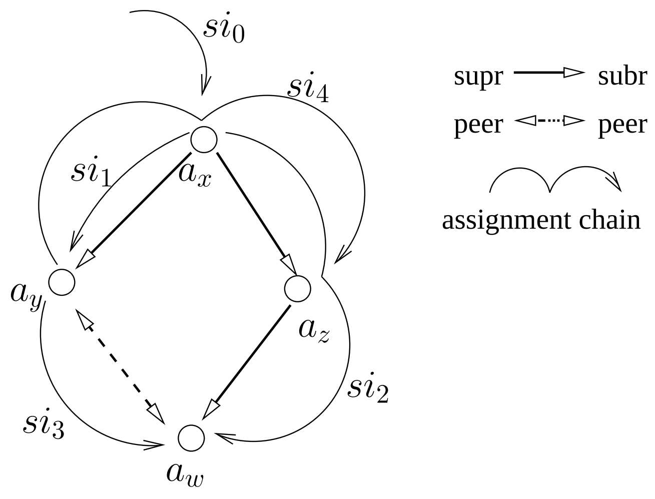
It is important to note that while the agents have their own individual benefit values, trying to selfishly increase their own benefit needn't necessarily lead to an increase in the overall benefit of the organisation because an agent's actions affect the other agents. For example, an agent can perform allocations quickly by having just one subordinate and delegating most tasks to that agent (thus not causing any loss to its own benefit), but that subordinate agent may be overloaded as a result, leading to a significant decrease in its benefit and that of the organisation. Therefore, the agents, being cooperative, need to maximise the organisation benefit, but they do not possess the complete information about the load and benefits of the other agents that contribute to the organisation benefit.

In order to better explain this evaluation process, we now present an illustrative example.

3.3.1 Example



(a) Task t_0



(b) Organisation

FIGURE 3.6: Assignment of service instances among the agents

Let us evaluate the cost and benefit values for agent a_x of the organisation in Fig 3.4 when it is performing the service instances belonging to the task t_0 in Fig 3.2 (reproduced in Fig 3.6(a)). Also, Fig 3.6(b) shows how the service instances are eventually distributed among the agents in the organisation. For ease of calculation, we assign values to the coefficients as:

$$C = 0.25 \qquad and \qquad M = 0.50 \tag{3.11}$$

Let us also assume the capacity of a_x to be $L_x = 2$. At the second time-step, having decided to execute si_0, agent a_x has to allocate the dependencies which are si_1 and si_2 to other agents.

It first allocates si_1 to a_y after traversing through its subordinates. This would take $2M$ load because it has two subordinates. Since this load is less than the capacity L_x, it then allocates si_2 in the same way to a_z again using up $2M$. Thus, $m_{0,x} = 4$ as the sum of the relations considered is 2 for each of the two service instances. Therefore, the total load on a_x, in this time-step, is (from Eqn 3.7):

$$l_x = 0 + M * 4 = 0.5 * 4 = 2$$

Also, a_x has to send two assignment messages, one to a_y and the other to a_z, informing them about the allocations. Therefore, the cost because of a_x is (from Eqn 3.5):

$$cost_x = C * 2 = 0.25 * 2 = 0.50$$

Similarly, the benefit from a_x is obtained from Eqn 3.8

$$benefit_x = 0 - 0 = 0$$

This is because, a_x did not execute any service instance in this time-step ($T_{x_E} = \emptyset$) and there are no tasks waiting for execution or allocation either ($T_{x_W} = \emptyset$). Since, all the other agents are inactive in this time-step (they haven't received any tasks yet), the overall cost of the organisation is (from Eqn 3.6):

$$cost_{ORG} = cost_x + cost_y + cost_z + cost_w = 0.50 + 0 + 0 + 0 + 0 = 0.50$$

Similarly, the benefit of the organisation is (from Eqn 3.9):

$$benefit_{ORG} = benefit_x + benefit_y + benefit_z + benefit_w = 0 + 0 + 0 + 0 = 0$$

Let us now consider another time-step and focus on agent a_w. Let us assume that the capacity of a_w is $L_w = 3$. Similarly, let service instances si_2 and si_3 have the following values:

$$si_2 = \langle s_2, 3, 10 \rangle \qquad and \qquad si_3 = \langle s_0, 1, 17 \rangle$$

Also, let us assume that a_w is executing service instance si_2 in this time-step, while si_3 is waiting for execution as there is not enough available capacity to execute si_3 while si_2 is being executed (since $3 - 3 < 1$). Now, as the agent does not send any messages, its cost (from Eqn 3.5):

$$cost_w = 0$$

However, the benefit from a_w depends on the service instances being executed ($T_{x_E} = \{si_2\}$) and service instances waiting for execution or allocation ($T_{x_W} = \{si_3\}$) as follows (from Eqn 3.8):

$$\mathit{benefit}_w = p_2 - p_3 = 3 - 1 = 2$$

where p_2 and p_3 are the computational rates of si_2 and si_3 respectively.

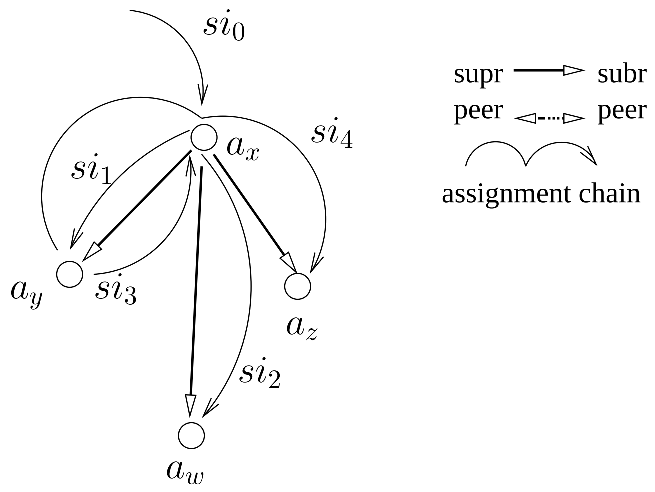


FIGURE 3.7: Another organisation and the task allocations

Now, let us consider a different organisation structure in Fig 3.7 executing the same task as the earlier one. In this case, a_x has the other three agents as its subordinates. Therefore, a_x allocates si_1 to a_y and si_2 to a_w. Also, a_y allocates si_3 to a_x. However, a_y has to delegate si_4 indirectly to a_z via a_x. The eventual allocations of the service instances across the organisation are also shown in the figure. Even in this case, let us evaluate the cost and benefit values for agent a_x in the second time-step. Similar to the earlier case, a_x has to assign service instances si_1 and si_2 to its subordinates. However, in this case, it has three subordinates. Therefore, allocating si_1 to a_y requires $3M$ load, as it has to traverse through all the three subordinates. Hence, the assignment process of si_1 eats up $3M = 3 * 0.5 = 1.5$ of the capacity L_x, which is 2. After this, a_x tries to assign s_2 but its capacity gets filled as soon as it checks its first subordinate a_y (which takes M load, that is 0.5). Therefore, the assignment process of si_2 will be continued in the next time-step. So, si_2 goes into the waiting list T_{x_W}.

Now, a_x also sends a message to a_y informing it about the assignment of si_1. Therefore, its cost is:

$$\mathit{cost}_x = C * 1 = 0.25$$

However, the benefit of a_x will be affected by the waiting service instance si_2 whose computational rate has to be subtracted. Therefore:

$$\mathit{benefit}_x = 0 - p_2 = -3$$

where p_2 is the computational rate of si_2, as stated earlier.

Similarly, for this case also, let us again consider a_w at the same time-step as in the earlier case, when it is executing si_2. In contrast to the earlier case, a_w has not been assigned si_3 (which has been assigned to a_x instead). Therefore, the waiting task set of a_w, $T_{wW} = \emptyset$ and hence, the benefit will not be reduced (unlike earlier). Thus, benefit of a_w is:

$$benefit_w = p_2 = 3$$

3.4 Summary

In this chapter, we introduced our organisation model by presenting our representation of tasks and organisations. The tasks are made up of service instances, each of which specify the particular service, the computational rate and the time-steps required. The organisation consists of agents providing services and having computational capacities. The relationships between the agents could be pure acquaintance, peer or superior-subordinate. The relations of the agents determine what service information is held by the agents about the other agents and how to allocate service instances to them. We also presented the coefficients that affect the environment (communication cost and management load) and the functions for calculating the cost and benefit of the organisation, thus enabling us to evaluate the efficiency of an organisation. Thus, we addressed the two aims of this chapter, which also form our first two requirements in Section 1.1, that involve developing a model for problem-solving agent organisations and a mechanism for its evaluation.

The next chapter presents our adaptation method which is based on the organisation framework developed in this chapter. The following chapter discusses the experiments conducted on this adaptation technique in order to evaluate its effectiveness, and analyses the results we obtained.

Chapter 4

Decentralised Structural Adaptation

The third objective presented in Section 1.1 requires the development of a completely decentralised agent-based structural adaptation approach for improving the efficiency of a problem-solving agent organisation. It suggests that the reorganisation technique should be employable by any agent in the organisation, at any time. Given this, this chapter attempts to satisfy this requirement by presenting just such a reorganisation mechanism.

In more detail, the first section of this chapter proceeds by examining the types of strong self-organisation (see Section 2.2.1) that can be present in problem-solving agent organisations, particularly elaborating on structural reorganisation. The next section describes our decentralised structural adaptation method using the organisation model presented in Chapter 3 as the application scenario. It first discusses the constraints placed on the model and then explains our reorganisation method. The final section summarises the chapter. The notation introduced in this chapter is also explained in the glossary in Appendix A.

4.1 Reorganisation Methods

As discussed in Section 2.2, there are several methods for self-organisation in agent systems, some of which are applicable to the adaptation of problem-solving agent organisations. A few of them work by changing the configuration of the organisation; as per our requirements, we also aim to continuously improve the efficiency of the organisation (see Section 3.3) by changing the configuration of the organisation (however, as discussed in Section 2.2.3, our requirements differ from the current approaches in many respects). This will entail modifying the agents and/or their interactions (the organisation's structure). Towards this end, we can identify three major types of adaptations and explain them using our formal organisation model. Specifically, adaptation that seeks to improve the efficiency of the organisation can be achieved by:

1. Adding and removing the agents in the organisation (i.e changes to the set of agents A).
2. Modifying the properties of the agents; that is agents obtaining new service abilities and losing old ones (changes to the service sets $\{s_i...\}$ of the agents A) and/or changing their optimal capacities (modifying L_x).
3. Modifying the organisation structure thereby giving agents different superiors, subordinates and peers (i.e changes to G).

We shall examine, in detail, each of these adaptation types in the rest of this section.

4.1.1 Changing the Set of Agents

Agents can be created and removed from the organisation depending on the variations in the demand for the services of the agents (as specified by the tasks). Thus, this adaptation method works by spawning clones of over-burdened agents to reduce the load on them, and terminating agents that are not being used sufficiently. Thus, both these steps may improve the efficiency of the organisation. In this context, OSD (see Section 2.2.3) provides a suitable mechanism for implementing this kind of reorganisation. A similar method presented by Hoogendoorn (see Section 2.2.3), involves duplicating the agents to ease the pressure on the system. When applied to our model, these methods involve adding and removing elements from the agent set A, along with adding and deleting the structural links associated with these agents from the organisation graph G.

4.1.2 Changing the Properties of the Agents

Instead of spawning and terminating agents as suggested above, agents could individually reorganise by changing the services they provide to meet the changing requirements. For example, Klein and Tichy (see Section 2.2.3) present a method in which agents change the services that they provide according to the rewards associated with the services. In this case, services whose requirements exceed the capacities of the agents providing them have positive rewards and vice versa. Also, the value of the reward is high if the unmet demand for the service is high and so on. Thus, this method helps in improving the performance of the organisation as efficiency is dependent on how closely the load matches the capacities of the agents. Formally, this method entails changes to the elements of A. More specifically, it involves changing the set of services $\{s_j \in S\}$ of the agents (a_x, a_y and so on) of A.

On similar lines, the computational capacities of the agents can be varied to match the load on them. This method involves modifying the L_x parameter of the respective agent a_x.

4.1.3 Changing the Structure of the Organisation

Changing the interactions between the agents by modifying the structure is also an effective means of achieving reorganisation as shown in some scenarios in Section 2.2.3. In our organisation model, as agents are performing services and finding other agents to allocate the prerequisite service instances, they may realise that they have a large amount of a particular kind of interaction with some of the agents. Thus, forming relations (in the control graph) with these agents (if not already present) may make the organisation more efficient by resulting in shorter assignment chains leading to lower cost and load on the agents. Similarly, dissolving existing relations with agents, with whom little or no interaction is taking place, will also improve the efficiency by reducing the management load on the agents as the agents will have fewer relations to consider while allocating service instances.

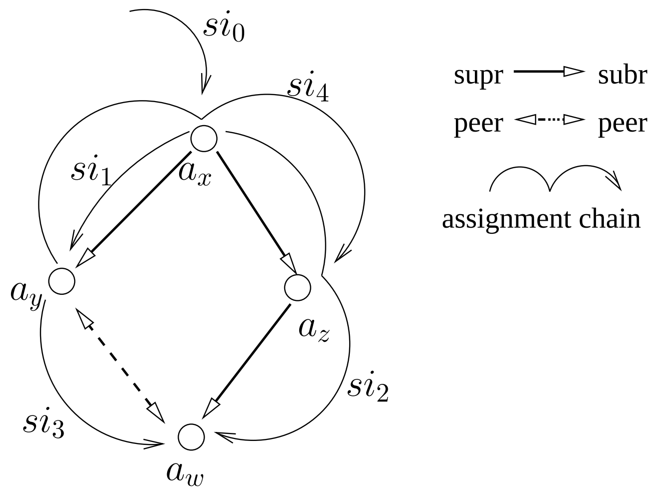


FIGURE 4.1: Organisation before adaptation

For instance, considering the earlier example of the sample organisation in Fig 3.4 executing the task in Fig 3.2, the allocation of service instances across the agents is as seen in Fig 4.1 (also explained in Section 3.2.2). We see that service instance si_2 is

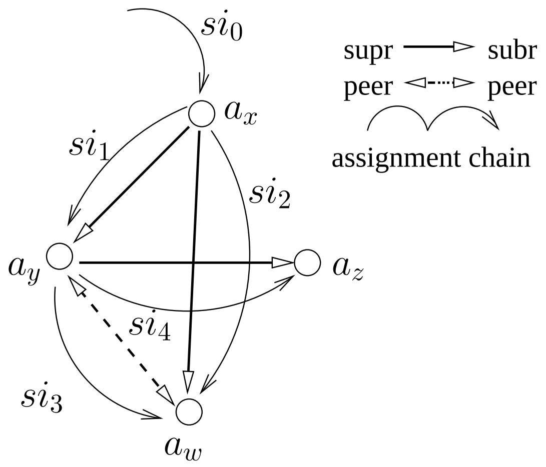


FIGURE 4.2: Organisation after adaptation

first allocated by a_x to a_z, who then reallocates it to a_w. Thus, the assignment process required two steps and involved an intermediate agent a_z, leading to some extra management load being put on a_z (due to the allocation process) and extra communication cost (since messages had to be indirectly passed forth and back between a_x and a_w via a_z). Similarly, the assignment of si_4 also required two assignment steps involving a_x as the intermediary.

However, had the organisation structure been different (see Fig 4.2), the load and cost could have been reduced leading to a better organisation efficiency. In detail, if a_x had a_w as its subordinate, then it could have directly allocated si_2 to a_w. This will involve just a single assignment step and one assignment message (unlike two steps and messages as required in the earlier case). Furthermore, if a_x and a_z did not have any relation, then the management load on a_x will be less as it will not have to consider a_z while performing allocations. Similarly, if a_y had a_z as its subordinate, then si_4 could have been directly allocated to a_z instead of indirectly through a_x. In this way, the performance of the organisation can be improved by modifying the organisation structure through changes to the agent relationships. This will involve changes to the organisation graph G.

Our motivation for developing self-organisation techniques in agent organisations is guided by the development of autonomic systems (see Section 2.2). In particular, agent based development of autonomic systems involves modelling the individual components as agents. Therefore, changing the characteristics of these components will not be possible on all occasions. For this purpose, we attempt to develop adaptation techniques that can be applied to organisations in which the agents and their internal characteristics cannot be changed. These kinds of organisations, in which the agents and their characteristics are fixed, are sometimes known as *closed organisations* (Lematre and Excelente, 1998). In a closed organisation, the first two types of adaptation (addition/deletion of agents and changing agent properties), as detailed in the previously, will have to be ruled out. Hence, we are primarily interested in the last type — changing the organisation structure. Therefore, while we intend to make our adaptation mechanism robust against changing agents and their characteristics also, we do not seek to develop methods that initiate changes to the agents or their characteristics as the means of reorganisation.

In summary, while the efficiency of the organisation can be improved by changing the agents, their internal characteristics or their external relations, we solely focus on the characteristics of the organisation like the agent relations. Next, we present our decentralised structural adaptation method.

4.2 Our Reorganisation Method

In this section, we first list the various constraints that we have placed on the organisation model for developing our reorganisation method. Following it, we describe the reorganisation strategy using a decision theoretic approach.

4.2.1 Constraints on the Model

Even though we will be essentially using the organisation framework described earlier (see Chapter 3), we need a slightly more restricted model and environment to permit us to focus solely on structural adaptation techniques. Therefore, we list the various constraints placed on the organisation and the environment for developing our reorganisation mechanism. Note that the formal notation used here is defined in Chapter 3

1. **Closed Organisation:** The agents present in the organisation are permanent. They cannot leave the organisation and, similarly, new agents cannot join. Formally, A is a static set.

2. **Invariant Agents:** The properties of the agents (i.e service set and computational capacity) are fixed. Therefore, every element a_x of A is constant. Moreover, the agents do not crash or fail in any manner. This constraint will be relaxed in the future work (see Section 6.1).

3. **Compulsory and Ordered Tasks:** All tasks must be accomplished by the organisation and the task processing should be initiated by the agent that the task arrives at, immediately (the next time-step) after the task is presented to the organisation. Therefore every task t_i belonging to the task set T needs to be completed by the organisation.

4. **Selfless Agents:** The agents are only interested in reducing the cost and increasing the benefit of the organisation and not their personal costs or benefits. Therefore the agents only aim to to reduce $cost_{ORG}$ and increase $benefit_{ORG}$ but not $cost_x$ or $benefit_x$.

5. **Agent Communication:** A message can be passed only between two agents (no broadcast is permitted). Also, any message costs C irrespective of its content or length. As previously detailed in Section 3.2.2, the three types of messages are — (i) assignment, (ii) acknowledgement and (ii) result of a service instance.

6. **Benevolent and Knowledgeable Agents:** Agents accept, without exception, any service instance assigned to them by the other agents. Also, an agent is acquainted with all other agents in the organisation. Thus, the acquaintance graph is completely connected.

7. **Reorganisation Cost:** The reorganisation cost associated with forming or dissolving the relations does not depend on the agents or the type of the relation but only the reorganisation cost coefficient. This reorganisation cost coefficient is denoted by R and formally represents the cost to the environment for forming or dissolving a relation between two agents.

8. **Reorganisation Load:** The process of deliberating about changing the relations with other agents is assumed to put no computational load on the agents and can be carried out in parallel to the actual task processing. Similarly, the non-task related messages that might be passed while deciding about reorganisation are also considered to not cause any communication cost to the organisation. Therefore $cost_x$ and l_x of an agent a_x are not affected by the reorganisation process.

By imposing these restrictions on the framework, we highlight the affect of the structure on the performance of the organisation through negating the unrelated factors like unreliable systems or network, external agents and so on. Therefore, our structural adaptation mechanism focuses on improving the performance of the organisation in terms of its efficiency. Against this background, we proceed to explain our reorganisation mechanism.

4.2.2 Our Reorganisation Approach

The aim of our reorganisation method is to determine and effect changes in the organisation structure to increase its efficiency (see Section 3.3). Our method is based on the past interactions of the agents. We use only the past information of the individual agents because we assume the agents do not possess any information[1] about the tasks that will come in the future. Specifically, agents use the information about all their past allocations to evaluate their relations with their subordinates, superiors, peers and pure acquaintances. This evaluation is based on the possible increase or decrease of the overall load and cost in case a subordinate or peer relation had previously existed (a pure acquaintance had been a subordinate or a peer and so on), the relation had been different (say, had a peer been a subordinate) or the relation hadn't existed (a peer or a subordinate had only been a pure acquaintance and so on). For example, an agent a_x evaluates its relation with its subordinate a_y on the basis of the number of its service instance dependencies that have been executed by a_y. More specifically, a_x assumes that had a_y not been its subordinate, then all its delegations to a_y would have gone indirectly via some intermediary agents. Therefore, a_x will check whether the possible reduction in its own load had a_y not been a subordinate (because one less subordinate will lead to a lower management load during allocations) is more than the possible increase in the load and cost across the organisation (due to the resultant longer assignment chains).

In more detail, we formulate our reorganisation method using a decision theoretic approach since it provides us with a simple and suitable methodology for formally representing the choices available to a decision maker, thus enabling it to make the right selection. Since, our reorganisation method involves agents evaluating and changing their relations, it is befitting to represent our method in terms of actions and utilities as specified by decision theory. We denote the actions of an agent as establishing or dissolving relations with other agents. Considering our model, there are two possible relations between any two agents— peer or authority (assuming that the agents are always acquaintances of each other). Therefore, four atomic actions are possible:- $E = \langle$ form_subr, rem_subr, form_peer, rem_peer $\rangle$. The actions are mutually exclusive and can be performed if the relation between the agents is in the requisite state (explained later). Moreover, each of these actions has to be jointly performed by the two agents involved in forming/removing the relationship link. Furthermore, the actions are deterministic (there is no uncertainty regarding the outcome of an action which is the formation or deletion of a link; only the utility of the outcome is not pre-determined). Therefore, the agents have a value function (also called an ordinal utility function) which they attempt to maximise.

Since our environment is characterised by various factors like the communication cost, the load on the agents, and so on, the value function will have multiple attributes to represent these different factors. In terms of two agents a_x and a_y jointly deliberating an action, we list the five attributes that will constitute the value function:

[1]We are only focusing on generic reorganisation methods that are not dependent on any extra knowledge about the task environment other than the tasks that have already been processed.

1. change to the load on a_x
2. change to the load on a_y
3. change to the load on other agents
4. change in the communication cost of the organisation
5. reorganisation cost incurred by the organisation for taking the action

We selected this set of five attributes because they cover all the factors affecting the efficiency, but at the same time, can be calculated independent of each other from the history of task assignments. Therefore, this set of attributes exhibits mutual preferential independence (MPI). That is, while every attribute is important, it does not affect the way the rest of the attributes are compared. Therefore, the value function can be represented as simply a sum of the functions pertaining to the individual attributes. That is, it is an additive value function of the form:

$$V = \delta load_x + \delta load_y + \delta load_{OA} + \delta cost_{comm} + cost_{reorg} \tag{4.1}$$

Against this background, we discuss the actions and states of the agents. For every pair of agents, the relation between them has to be in one of the states— purely an acquaintance relation, peer relation or superior-subordinate relation. For each of these states, the possible choices of action available to the agents are:

1. a_y **is a pure acquaintance of** a_x**:** (i) $form\_peer_{x,y}$; (ii) $form\_subr_{x,y}$; (iii) *no_action*.
2. a_y **is a subordinate of** a_x**:** (i) $rem\_subr_{x,y}$; (ii) $rem\_subr_{x,y} + form\_peer_{x,y}$ (to change to a peer relation); (iii) *no_action*.
3. a_y **is a peer of** a_x**:** (i) $rem\_peer_{x,y}$; (ii) $rem\_peer_{x,y} + form\_subr_{x,y}$ (to change to a subordinate relation); (iii) *no_action*.
4. a_y **is a superior of** a_x**:** (i) $rem\_subr_{y,x} + form\_subr_{x,y}$; (ii) $rem\_subr_{y,x} + form\_subr_{x,z}$ (where a_z is a (indirect) superior[2] of a_y); (iii) *no_action*.

The possible actions for transitions between the various states are further illustrated by Fig 4.3.

Thus, three actions are possible in any state. For example, the $form\_subr_{x,y}$ action denotes that a_x and a_y take the action of making a_y a subordinate of a_x. In this way,

[2]when a_z is an indirect superior of a_x via a_y, it is not possible for a_x to have a_z as its subordinate (since cycles of authority links in the control graph are not permitted). Hence, making a_z a subordinate entails dissolving its relation with its immediate superior in the authority chain which is a_y.

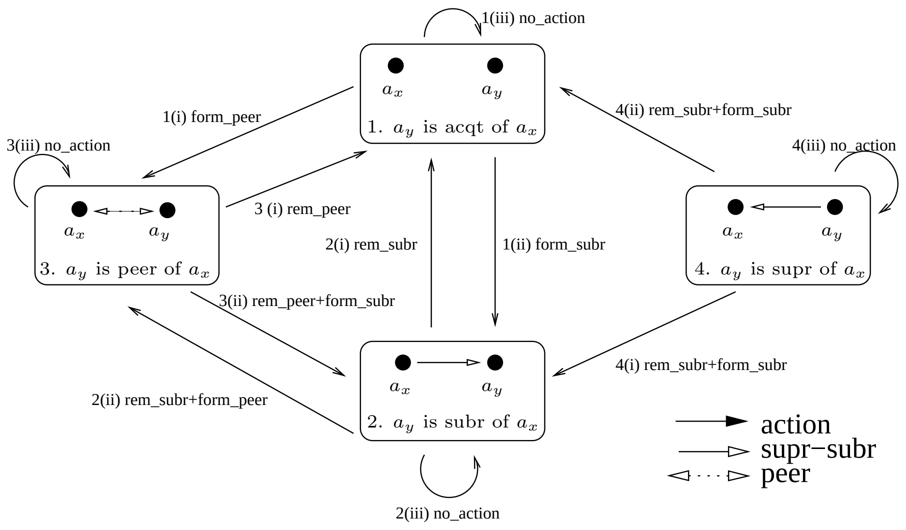


FIGURE 4.3: State transition diagram

depending on the state, the agents jointly calculate the expected utilities for the possible actions based on the value function and choose the action giving the maximum expected utility in accordance with standard decision theory. The evaluation for *no_action* will always be 0 as it does not result in any change to the expected load or cost of the organisation. The evaluation for the rest of the actions is obtained from the value function (Eqn 4.1). The evaluation for the composite actions (*rem_subr* + *form_peer*, *rem_peer* + *form_subr* or *rem_subr* + *form_subr*) for transition between two relations will simply be the sum of the individual evaluations of the comprising actions. Moreover, since any action will be taken jointly by the two agents involved, even the evaluation of the value function is jointly conducted by the agents with each of them supplying those attribute values that are accessible to them.

Against this background, we move on to discuss the calculation of the value function for the various actions by detailing how the individual attributes are computed.

4.2.3 Value Function Calculation

The agent's value function is the sum of the values of the attributes as shown in Eqn 4.1. To represent the efficiency condition of the organisation (Eqn 3.10), a reduction in cost or load (excess load adversely affects the benefit) is considered to add positively to the value and vice versa. Table 4.1 lists the calculation of the attributes of the value function for each of the four atomic actions. This table uses the additional notation (in addition to that defined in Chapter 3) that we describe below:

Assignment: By assignment of a service instance to an agent, we mean that the agent has been allocated that service instance. It must accomplish that service instance by either executing it itself or by assigning it again to some other agent. Formally, when a service instance si_i is assigned to agent a_x, then a_x is considered an *assigned agent* for si_i.

Delegation: By delegation of a service instance to an agent, we refer to the fact that the agent is executing the particular service instance by itself (without reassigning it to someone else). Formally, when an agent a_x executes a service instance si_i, it is considered as the *delegated agent* for si_i. Thus, there could be several assigned agents for a particular service instance, but only one delegated agent.

1. $Asg_{x,y}$**:** The number of service instances assigned by an agent a_x to a_y. Assignment of a service instance si_i by a_x to a_y means that a_x required that si_i be executed (was assigned to a_x or forms dependency of an assigned service instance of a_x) and it reallocated si_i to a_y. Thus a_y will have to be a subordinate, peer or a superior of a_x. Also a_y need not necessarily execute si_i itself, it could reassign it to one of its own subordinates, peers or superiors.

2. $Del_{x,y}$**:** The number of service instances delegated by an agent a_x to a_y. Delegation of a service instance si_i by a_x to a_y means that a_x is the agent that first required that si_i be executed (as it formed a dependency of a service instance being executed by a_x) and a_y is the agent that finally executed si_i (that is, a_y is the delegated agent). Note that, a_y may just have a pure acquaintance relationship with a_x. The delegation is always achieved through one or more assignments.

3. **C:** Communication cost coefficient.

4. **M:** Management load coefficient.

5. **R:** Reorganisation cost coefficient.

6. n^{tot}**:** The total number of time-steps that have elapsed since the beginning of the run.

7. $n^{subr}_{x,y}$**:** The number of time-steps that a_x and a_y had a superior-subordinate relation (that is, time-steps that $\langle a_x, a_y, Supr\rangle \in G$) . Likewise, $n^{peer}_{x,y}$ denotes the amount of time that a_x and a_y had a peer relation.

8. $filled_x(n)$: The number of time-steps out of the total time denoted by n that a_x had waiting tasks ($T_{xW} \neq \phi$; capacity being completely filled by load). The variable n can represent the total time passed (n^{tot}) or the time duration that a_y was its peer ($n^{peer}_{x,y}$) and so on.

9. $Asg_{x,subr}$**:** The number of service instances that have been assigned by a_x to any of its subordinates. Likewise, $Asg_{x,peer}$ and $Asg_{x,supr}$.

10. $Asg_{x,tot}$**:** The total number of service instances that have been assigned by a_x to other agents. Therefore, $Asg_{x,tot} = Asg_{x,subr} + Asg_{x,peer} + Asg_{x,supr}$.

11. $Asg^{LOAD}_{x,y}$: The management load added onto a_y because of assignments from a_x (the count of these assignments is denoted by $Asg_{x,y}$ as stated above).

12. $IA_{x,y}$: The total number of times, other agents (intermediate agents) were involved in the delegations of service instances by a_x to a_y. Therefore,

$$IA_{x,y} = \sum_{i}^{Del_{x,y}} count^{i}_{OA}$$

where $count^{i}_{OA}$ is the number of other agents involved in the delegation of si_i from a_x to a_y.

13. $IA^{COST}_{x,y}$: The communication cost due to the delegations from a_x to a_y. For every agent in $IA_{x,y}$, a cost of $2C$ is added because a message is once sent forward and once back. Therefore, $IA^{COST}_{x,y} = IA_{x,y} * 2 * C$.

14. $IA^{LOAD}_{x,y}$: The overall management load put on all the intermediate agents involved in the delegations from a_x to a_y (that is, $Del_{x,y}$). The load values are reported back to a_x along with the assignment information (the assignment message). If an intermediate agent has available capacity (no waiting tasks), it will report a 0 load value for that delegation. Otherwise, the agent will report the actual management load that was put on it due to that service instance assignment.

Following this notation, Table 4.1 lists out how the five attributes are calculated for each of the four basic actions. The last column in the table denotes which agent will be performing the calculation for that particular attribute.

More specifically, Table 4.1(a) represents the $form\_subr_{x,y}$ action. When two agents, a_x and a_y, need to evaluate whether forming a superior-subordinate relation will be beneficial, they need to estimate the utility of taking such an action. As stated earlier, this utility is obtained from the value function (Eqn 4.1). However the calculation of the attributes, that make up the value function, varies according to the action for which they are being calculated. Therefore, Table 4.1(a) shows how the attributes are calculated for estimating the increase in value by taking the $form\_subr_{x,y}$ action. Similarly, Table 4.1(b) refers to $rem\_subr_{x,y}$, Table 4.1(c) refers to $form\_peer_{x,y}$ and Table 4.1(d) refers to $rem\_peer_{x,y}$ action.

TABLE 4.1: Attribute functions for the reorganisation actions

(a) Action $form\_subr_{x,y}$ between agents a_x and a_y (with a_x as the superior)

	Attribute	Function	Agent
(i)	$\delta load_x$	$-Asg_{x,tot} * M * filled_x(n^{tot})/n^{tot}$	a_x
	The management load that will be added on a_x due to an additional subordinate. This is estimated by counting all the assignments that a_x had to perform until now and adding a load of M for each of those. This is because, if a_y had been a subordinate of a_x from the beginning, then it will have been considered for each of the assignment of a_x causing an extra load of M for every such instance. This value is then multiplied by a factor which represents how often this increased load will affect the benefit. This factor is the amount of time-steps that a_x had waiting tasks divided by the total time. The reasoning is that the increased load will affect the benefit of a_x only when its capacity is filled.		
(ii)	$\delta load_y$	$-Asg^{LOAD}_{x,y} * filled_y(n^{subr}_{x,y}) * n^{tot}/(n^{subr}_{x,y})^2$ or 0 if $n^{subr}_{x,y} = 0$	a_y
	The management load that will be added on a_y due to possible assignments from a_x. It is estimated by considering the load on a_y due to a_x when it was a subordinate of a_x previously and multiplying it with the fraction of time that a_y had waiting tasks while in the relation and dividing it by the fraction of total time that the relation existed. Thus, the calculated value is normalised to correspond to the total time and not the relation time.		
(iii)	$\delta load_{OA}$	$IA^{LOAD}_{x,y}$	a_x
	The reduction in the management load on the intermediate agents that had been involved in the delegations to a_y by a_x. As this load value is sent by an intermediate agent only if it has waiting tasks, no time factor like the one in a(i) is required.		
(iv)	$\delta cost_{comm}$	$IA^{COST}_{x,y}$	a_x
	The reduction in communication cost that was associated with the delegations to a_y by a_x.		
(v)	$cost_{reorg}$	$-R$	constant
	This is the reorganisation cost associated with forming or dissolving a relation. Though, it is a one time cost, it's affect is spread over the same time window that the rest of the evaluations are valid.		

(b) Action $rem\_subr_{x,y}$ between agents a_x and a_y (a_x is the superior)

	Attribute	Function	Agent
(i)	$\delta load_x$	$Asg_{x,tot} * M * filled_x(n^{subr}_{x,y})/n^{subr}_{x,y}$	a_x
	The management load on a_x that will be reduced because a_y will no longer be a subordinate. The assignments count for the complete time is obtained and then multiplied by the fraction of time that a_x had waiting tasks while in the relation so that the load values correspond to not just the total time but also reflect the factor by which they will affect the benefit.		
(ii)	$\delta load_y$	$Asg^{LOAD}_{x,y} * filled_y(n^{subr}_{x,y})/n^{subr}_{x,y} * n^{tot}/n^{subr}_{x,y}$	a_y
	The management load on a_y that will reduce because it will no longer get direct assignments from a_x. It is multiplied by the fraction of time that it had waiting tasks while in the relationship, similar to a(ii). As in that case, since the load value correspond to the relation time only, it is divided by the fraction of the time the relation existed so that the calculated value corresponds to total time.		
(iii)	$\delta load_{OA}$	$-IA^{LOAD}_{x,y} * n^{tot}/(n^{tot} - n^{subr}_{x,y})$	a_x
	The management load that will be put on other agents for delegations to a_y from a_x if it were not a subordinate of a_x. It is estimated by considering the load on intermediate agents, as in a(i), when the relation didn't exist and dividing by the fraction of time that the relation didn't exist so that the load values correspond to the total time.		
(iv)	$\delta cost_{comm}$	$-IA^{COST}_{x,y} * n^{tot}/(n^{tot} - n^{subr}_{x,y})$	a_x
	The addition to the communication cost associated with the delegation multiplied by the time fraction as in b(iii).		
(v)	$cost_{reorg}$	$-R$	constant
	Similar to a(v).		

Note that the negative sign, whenever present in the attribute calculations, denotes that the value represents an increase in the load or cost. Also, the attributes are calculated such that the resultant value for any attribute and any action is always normalised to the total elapsed time of the simulation. Therefore, the utility of one action can be compared directly against that of another action, thus allowing the agents to make a decision. For this reason, the load and cost values are multiplied by time fractions such that the final value always corresponds to the the total time elapsed. The value function represents the expected change in the load and cost of the organisation if the particular action is taken. Therefore, the intuition behind the attribute calculations is that the past assignments and delegations between the two agents (a_x and a_y) will provide a reasonable indication whether forming or dissolving their relation will reduce the overall load and cost of the organisation and by how much.

(c) Action $form\_peer_{x,y}$ between agents a_x and a_y

Attribute		Function	Agent
(i)	$\delta load_x$	$-(Asg_{x,peer} + Asg_{x,supr}) * M * filled_x(n^{tot})/n^{tot}$	a_x
The management load that will be added on a_x due to an additional peer. It is multiplied by the time factor similar to a(i).			
(ii)	$\delta load_y$	$-(Asg_{y,peer} + Asg_{y,supr}) * M * filled_y(n^{tot})/n^{tot}$	a_y
Similar to c(i).			
(iii)	$\delta load_{OA}$	$IA^{LOAD}_{x,y} + IA^{LOAD}_{y,x}$	a_x and a_y
The reduction in the management load on intermediate agents involved in the delegations from a_x to a_y and from a_y to a_x.			
(iv)	$\delta cost_{comm}$	$IA^{COST}_{x,y} + IA^{COST}_{y,x}$	a_x and a_y
The reduction in the communication cost associated with the delegations in c(iii).			
(v)	$cost_{reorg}$	$-R$	constant
Similar to a(v).			

(d) Action $rem\_peer_{x,y}$ between agents a_x and a_y

Attribute		Function	Agent
(i)	$\delta load_x$	$(Asg_{x,peer} + Asg_{x,supr}) * M * filled_x(n^{peer}_{x,y})/n^{peer}_{x,y}$	a_x
The management load on a_x that will be reduced because a_y will no longer be a peer. Multiplied by the time fraction similar to b(i).			
(ii)	$\delta load_y$	$(Asg_{y,peer} + Asg_{y,supr}) * M * filled_y(n^{peer}_{y,x})/n^{peer}_{y,x}$	a_y
Similar to d(i).			
(iii)	$\delta load_{OA}$	$-(IA^{LOAD}_{x,y} * n^{tot}/(n^{tot} - n^{peer}_{x,y}) + IA^{LOAD}_{y,x} * n^{tot}/(n^{tot} - n^{peer}_{y,x}))$	a_x and a_y
The management load that will be added to other agents for delegations from a_x to a_y and from a_y to a_x. It is estimated in the same way as b(iii).			
(iv)	$\delta cost_{comm}$	$-(IA^{COST}_{x,y} * n^{tot}/(n^{tot} - n^{peer}_{x,y}) + IA^{COST}_{y,x} * n^{tot}/(n^{tot} - n^{peer}_{y,x}))$	a_x and a_y
The increase to the communication cost because of the delegations. It is estimated in the same way as b(iv).			
(v)	$cost_{reorg}$	$-R$	constant
Similar to a(v).			

Using the value function, every pair of agents jointly evaluates the utility for taking any of the possible actions (depending on their relation) towards changing their relation, at every time-step (though, newly formed relations are allowed to stabilise and are re-evaluated only after a prefixed time interval). Here, joint evaluation means that each agent supplies only some of the attribute values (as evident from the table), but the resultant utility for the action is applicable to both; thus eliminating any possible conflicts. Hence, this continuous adaptation in the relations helps in the better allocation of service instances amongst the agents as they will maintain relations with only those agents with whom they require frequent interactions.

Additionally, when an agent has two or more service instances of a particular service waiting for execution (because of filled capacity), it will search among its acquaintances for agents capable of performing that highly demanded service and will specifically calculate the utility of forming a superior-subordinate relation with these agents. If any of these prospective subordinates is already a peer, or a superior (including even an indirect superior, farther up in the authority chain), the agents will calculate the value function for the possible combination of actions permitted by their state, as earlier shown in Fig 4.3. However, while the agent calculates the utility using the table, it will also include the possible gain from the decrease to its own load resulting from the delegation of its future waiting load (when the relation is formed) to the proposed subordinate. Thereby, overloaded agents can form subordinate relations with other similarly capable agents leading to a more equitable load distribution across the organisation.

We further illustrate our reorganisation method with the help of an example.

4.2.4 Example

Consider the earlier example of the sample organisation in Fig 3.4 executing the task in Fig 3.2. The allocation of service instances across the agents is reproduced in Fig 4.4 (explained in Section 3.2.2). Let us look at agent a_x evaluating its relations at time-step 5, after the task has been completely allocated across the agents. Firstly, consider that a_x and a_z are evaluating their relation. According to Fig 4.3, since a_z is a subordinate of a_x, their relation is in state 2 with three possible actions — (i)$rem\_subr_{x,z}$, (ii) $rem\_subr_{x,z} + form\_peer_{x,z}$ and (iii) no_action. Let us first focus on how they calculate the expected utility of $rem\_subr_{x,z}$ using Table 4.1(b).

Similar to the example in the previous chapter (Section 3.3.1), let the values of the coefficients be:

$$C = 0.25 \qquad and \qquad M = 0.5 \qquad and \qquad R = 0.10$$

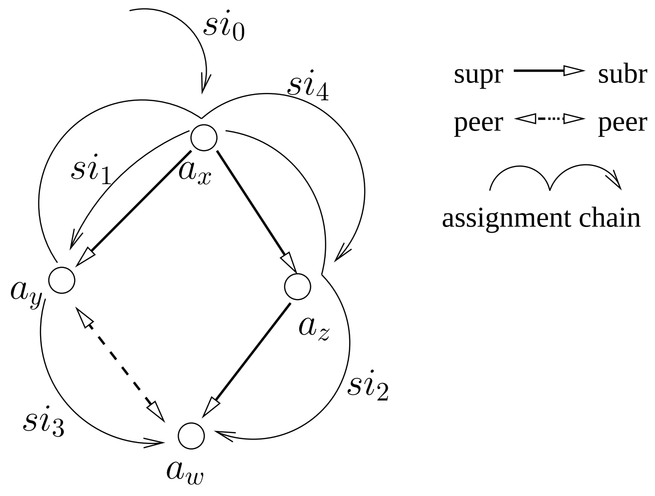


FIGURE 4.4: Allocation and execution of the service instances by the agents

As 5 time-steps have elapsed since the start, $n^{tot} = 5$. Also, we assume that all the relations existed, as shown, from the beginning. Therefore, the time-period for any of the existing relations (like $n^{subr}_{x,z}$) is also 5. Moreover, assume that the capacity of agent a_x was never filled over these 5 time-steps. Given this, let us look at the first attribute of the value function, $\delta load_x = Asg_{x,tot} * M * \mathit{filled}_x(n^{subr}_{x,z})/n^{subr}_{x,z}$. From the Fig 4.4, we see that a_x had to perform three assignments (si_1 to a_y, si_2 to a_z and si_4 also to a_z). Therefore, $Asg_{x,tot} = 3$. However, $\mathit{filled}_x(n^{subr}_{x,z}) = 0$ since a_x never had any waiting tasks over this time. Hence, $\delta load_x = 3 * 0.5 * 0 = 0$.

Similarly, looking at the second term $\delta load_z$, $Asg^{LOAD}_{x,z} = 0.5$ since a_z had to perform an assignment (of si_2) for a_x and it took up M load as a_z had only one subordinate (a_w). Let us assume that a_z's capacity was completely filled for the last time-step (when it started executing si_4). Therefore, $\mathit{filled}_z(n^{subr}_{x,y}) = 1$, while $n^{subr}_{x,z}$ and n^{tot} are 5 each. Hence, $\delta load_z = 1 * 1/5 * 5/5 = 1/5 = 0.20$

Now, looking at the third term, we find that $IA^{LOAD}_{x,z} = 0$, since there were no delegations from a_x to a_z (si_4 which is being executed by a_z was delegated by a_y to a_z, and not a_x which had only assigned it). As a result, $\delta load_{OA} = 0$. Similarly, $\delta cost_{comm} = 0$ too. Finally, $cost_{reorg} = -0.10$ as $R = 0.10$.

Having obtained the values of all the five attributes, the sum total of the expected utility of action $rem\_subr_{x,z} = 0 + 0.20 + 0 + 0 - 0.10 = 0.10$.

Next, we look at the second possible transition from this state which is $rem\_subr_{x,z} + form\_peer_{x,z}$. The evaluation for the *rem_subr* action has already been obtained. So, we focus on the *form_peer* action. The first term $\delta load_x = 0$ since a_x had not assigned any service instances to any peers or superiors ($Asg_{x,peer} + Asg_{x,supr} = 0$). Similarly, $\delta load_z = 0$ as a_z also did not assign to any peers or superiors. The third and fourth terms,

$\delta load_{OA}$ and $\delta cost_{comm}$ are also 0 as above because there were no delegations between a_x and a_z (as described above). Adding all the terms, the expected utility of action $form\_peer_{x,z} = 0+0+0+0-0.10 = -0.10$. Therefore the overall utility of the composite action of removing the subordinate and forming the peer is: $0.10 + (-0.10) = 0$.

The evaluation for the third possible action (*no_action*) is 0 by default. So comparing the expected utilities of all the three possible transitions, agents a_x and a_z find that $rem\_subr_{x,z}$ provides the maximum utility (0.10) and therefore, they take that action. Thus, the superior-subordinate link between a_x and a_z is dissolved according to our method. It should be noted that the values calculated here are very low because very little time had elapsed since the beginning of the run, and these values will become larger as time goes on and more tasks are processed by the organisation.

Continuing the example, let us consider a_x and a_w evaluating whether they should change their acquaintance relation to a superior-subordinate or a peer relation according to the transitions from state 1 in Fig 4.3. Note that this evaluation is also taking place at time-step 5 as earlier. We first observe their $form\_peer_{x,w}$ evaluation.

Similar to the above *form_peer* calculation, $\delta load_x$ and $\delta load_w$ will be 0 as neither agent has assigned any service instances to peers or superiors. For the $\delta load_{OA}$ calculation, $IA^{LOAD}_{x,w} = 0$ even though there was a delegation from a_x to a_w (service instance si_2). This is because, the only intermediary agent a_z's capacity was not completely filled when it performed the delegation. Therefore, it reported a value of 0 to a_x along with the acknowledgement message. Also, $IA^{LOAD}_{w,x} = 0$ as a_w did not delegate any service instances to a_x. However, $IA^{COST}_{x,w} = 1 * 2 * C = 1 * 2 * 0.25 = 0.50$ because the count of intermediate agents ($IA_{x,w}$) is 1 (a_z). As a result, $\delta cost_{comm} = 0.50$ because the other term, $IA^{COST}_{w,x} = 0$. Overall then, the expected utility of action $form\_peer_{x,w} = -0 - 0 + 0 + 0.50 - 0.10 = 0.40$.

Similarly, calculating the expected utility of the action $form\_subr_{x,w}$ also results in 0.40. The utility of the third possible action, *no_action* is default to 0, so the agents a_x and a_z can choose arbitrarily between these two actions (in this case, let us say they chose *form_subr*).

Finally, a_x also evaluates its relation with a_y by looking at the possible actions, which are $rem\_subr_{x,y}$, $rem\_subr_{x,y} + form\_peer_{x,y}$ and *no_action*. Their calculation of the attributes, similar to the above, reveals an expected utility of -0.10 for $rem\_subr_{x,y}$ and -0.20 for $rem\_subr_{x,y} + form\_peer_{x,y}$. Therefore, they chose the third alternative *no_action* which just has an expected utility of 0. Thus, after a_x has re-evaluated all its relations, the structure of the organisation will be as seen in Fig 4.5. The same steps will be followed by the rest of the agents also, depending on their relations.

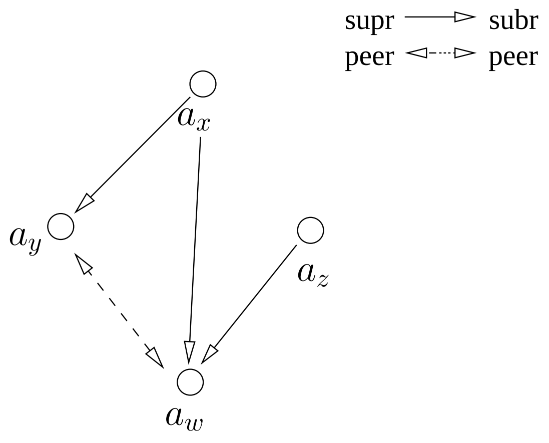


FIGURE 4.5: Organisation structure after a_x had reorganised

4.3 Summary

In this chapter, we first identified the three types of adaptation applicable to problem-solving agent organisations. These are creating/removing the agents, changing the agent properties and changing the structure of the organisation. Next, we listed the various constraints and assumptions placed on the organisation framework for developing our reorganisation method. Then, we presented our structural adaptation method using a decision theoretic approach in the form of actions and their expected utilities that will be used by all the agents in the organisation to modify their relations with other agents.

In conclusion, our reorganisation method guides the agent to form, change or dissolve relations with other agents in the organisation on the basis of their history of interactions. Thus, it will result in changes to the organisation structure in an attempt towards improving the efficiency of the organisation. Clearly, our method is completely decentralised as it will be adopted by all the agents in the organisation using only their local information. Moreover, it is used by the agents throughout their existence, thus making the adaptation a continuous process. Therefore, our method satisfies the properties of self-organisation that we outlined in Section 2.2.1. Moreover, the method is also in line with our third requirement, as specified in Section 1.1, referring to developing a decentralised agent-based structural adaptation approach.

The next chapter explains the experimental conditions used for evaluating our adaptation method and discusses the results obtained.

Chapter 5

Experiments and Results

This chapter discusses the various experiments conducted to evaluate the performance of the adaptation mechanism described in previous chapter. The first section of this chapter describes the experimental design and lists the various parameters of the simulation and the experimental variables under observation. The next section presents and discusses the results of the simulations. The final section summarises the chapter. This chapter satisfies the fourth and final requirement, stated in Section 1.1, which refers to testing and analysing the performance of our adaptation method.

5.1 Experimental Design

The purpose of the experiments is to evaluate the effectiveness of the structural adaptation mechanism developed for agent organisations. Since a standard is required for comparison, in addition to our method, we also consider four other procedures for modifying the organisational structure. These procedures constitute the control experiments. They are:

1. **Static:** A non-reorganising approach, the structure of the organisation is not modified during the simulation run.
2. **Random:** The agents randomly choose some of their relations for modification every time-step. The rate of change of relations is based on a probability which is inversely proportional to the number of relations they have. Thus, agents with fewer peers (similarly subordinates) will form more peers and vice versa. This probability was adjusted such that the overall number of changes in relations in a simulation run is roughly equal to that of our approach on an average so that the performance is not affected due to the aggregation of reorganisation cost.

3. **Central:** A central agent, which is external to the organisation, is responsible for the allocation of service instances. This agent has complete information about the organisation including the services, capacities and current load on the agents. When any agent needs to find another agent for allocating a service instance, it will assign it to the central agent, who will then delegate it to the most suitable agent (capable of the service and having the highest free capacity). Thus, every delegation is a fixed two step process with the central agent acting as an intermediary. Also, the organisation can be viewed as having a star structure in which all the agents are connected solely to this external central agent. Moreover, since this central agent is external, it is assumed to have infinite capacity and neither does it contribute to the load, benefit or cost calculations of the organisation.

4. **Oracle:** Instead of a single, central external agent as above, we assume that every agent has complete information about the organisation and so performs the best allocation of service instances possible. However in this case, the agents are assumed to not incur any load for their allocation actions ($M = 0$). Thus, as all the delegations are a one-step process without causing any management load, this approach acts as an upper bound for our evaluation.

We selected these four approaches because they sufficiently cover the main possibilities of reorganisation and task allocation mechanisms. Two of them (Random and Static) represent reorganisation (or the lack of) approaches, while the other two (Central and Oracle) represent complete-information based approaches for task allocations. Moreover, while Central represents a modified standard non-robust method and the Oracle procedure represents a clear upper bound, it is not obvious whether Random or Static is the lower bound and therefore we use both of them. For ease of reference, we shall call our adaptation procedure 'Smart'. Agents in the Smart, Static and Random procedures follow the same algorithm for task allocation as detailed in Section 3.2.2, while those in the Central and Oracle procedures follow their individual methods detailed as above. However, the agents in all the procedures are similar in all the other aspects (like task execution, communication, capacities and so on). We evaluate the effectiveness of the the procedures on the basis of the efficiency of the organisations which is determined by the average cost and benefit (see Section 3.3) of the organisation for a simulation run. Thus, $cost_{ORG}$ and $benefit_{ORG}$ are the two experimental data variables of interest. However, there are many other variables that need to be assigned values before running the experiments. Next, we discuss the values that are assigned to these parameters of the simulations.

5.1.1 Simulation Parameters

Simulation parameters are the attributes of the environment which are controlled according to the experiments. In our case, the simulation parameters include the set of services, set of agents, set of tasks, the initial organisation structure (at the beginning of simulation), the time period of the simulation and the environmental coefficients. While some of these parameters are assigned a constant value for all simulations, the others are varied across simulations. More specifically Table 5.1 lists the constant values or the fixed ranges assigned to some of the simulation parameters. The parameters listed in this table are applicable to all the simulations conducted. However, there are other parameters that are varied across simulations to observe their effect on the experimental variables ($cost_{ORG}$ and $benefit_{ORG}$). These are discussed below:

- **Distribution of services across agents:** After deciding the number of agents and services for a particular simulation, the next step is to distribute these services among the agents. The two extreme possibilities are: (i) each agent is capable of a unique set of services and (ii) all agents are capable of all services. To model this, the services are allocated to the agents on the basis of a probability called *service probability* (SP). That is, an agent a_x is allocated a service s_i with a probability SP. Thus, when SP is 0, every agent is capable of a unique service only (as every agent should offer at least one service and every service should be offered by at least one agent). When it is 1, every agent is capable of every service. Since the services are allocated on basis of a probability, there is always randomness in the way the services are allocated to the agents. In our experiments, we vary SP from 0 to 0.5 only (as we found that beyond 0.5, the agents are quite homogeneous and the organisation structures did not influence the performance significantly).

- **Similarity between tasks:** As discussed in Section 2.1.1, the tasks presented to the organisation over the period of a simulation run may be completely unrelated or somewhat repetitive in nature. In our model, similarity between tasks means that they may have some common service instances and dependency links. We control the similarity between the tasks belonging to the same simulation run on the basis of what we call *patterns*; stereotypical task components used to represent frequently occurring combinations of service instances and dependency links. Therefore, they are also composed of service instances, like tasks, but are generally smaller in size. So, instead of creating tasks by randomly creating dependency links D_i between the service instances, tasks can be constituted by connecting some patterns by creating dependency links between the service instances belonging to the patterns. In this way, the dependencies between the service instances may follow some frequent orderings (resulting from the dependencies internal to a pattern as the pattern may occur in several tasks) and some random dependencies (due to the dependencies created across the patterns). Thus, this method of generation enables us to control the similarity between the tasks using the *number of patterns* (NoP) as the parameter. In our experiments, we vary NoP from 1 (all tasks are composed of the same pattern repeated multiple times leading to a high degree of similarity) to 10 (several different patterns interlinked leading to low similarity across tasks). We limit at 10 because in our tasks, the maximum number of service instances is 20, and therefore, with too many patterns, there is no distinguishable occurrence of frequent dependencies and it is equivalent to having no patterns. Moreover, we also conducted a set of experiments with NoP set to 0 (tasks are completely dissimilar).

We consider that the similarity between the tasks is an interesting parameter to vary because it will provide us insights into how well our adaptation method performs in different kinds of task environments. This is an important factor for evaluation because our method is based on the past interactions between the agents which are, in turn, related to their past task allocations and executions. Moreover, presence of similarities between tasks is an existing phenomenon in the real world. When tasks of several kinds are received by a system, often, the tasks comprise of parts that are common across the different tasks. For example if the tasks are about constructing different cars, they will have some common sub-tasks like making a wheel, a windshield etc. Similarly, if the tasks are related to cooking, many of them may be composed of common sub-tasks like boiling potatoes, dicing carrots, marinating meat etc.

Against this background, we next discuss our hypotheses and that the experiments we carried out.

5.1.2 Hypotheses and Experiments

The controlled parameters of the environment which we predict will have a direct bearing on the performance are the ones described earlier in this section, namely, distribution of services across the agents and similarity of tasks. More specifically, we present two hypotheses about the factors influencing the effectiveness of structural adaptation:

1. The effectiveness of structural adaptation depends on the heterogeneity of the services of the agents. Specifically, structural reorganisation will be more effective on specialised agents than on homogeneous ones.

[1]We experimented with small tasks having a maximum of 10 service instances and large tasks with 50 service instances and found the results to be similar to having tasks with 20 service instances

Table 5.1: Values of the simulation parameters

Parameter	Symb	Value assigned	Justification
Number of agents	$\|A\|$	chosen from a uniform distribution between $4-20$	Organisations generally have at least 4 members. We chose 20 as the upper limit because we found that when number of agents is greater than 20, the simulations are considerably slower because of the constraint that all agents in the organisation are acquaintances of each other leading to $\theta(\|A\|^2)$ complexity for a single simulation time-step.
Initial organisation structure	G	randomly generated	Since we experiment with thousands of runs and hundreds of tasks in each run, the initial structure will not carry much significance.
Number of services	$\|S\|$	equal to $\|A\|$	The number of services possible is equal to the number of agents as the number of services matter only in terms of their distribution across the agents.
Computational rate of a service instance	p_i	chosen from a uniform distribution between $1-10$	This values determines the other values like capacity, so we decided to give it a standard range.
Capacity of an agent	L_x	chosen from a uniform distribution between $11-30$	Minimum is set to 11 so that the maximum computational rate is always less than the capacity. Maximum is set at 30 so that agents' capacities are filled up occasionally with 3 or 4 service instances.
Number of time-steps for a service instance	n_i	chosen from a geometric distribution (with p value set to 0.7) between $4-30$	There will be a large number of service instances with small time requirements (≤ 10) and some service instances with high time requirements. Thus, the task environment is quite dynamic but doesn't fluctuate too rapidly.
Arrival agent of a task	a_x	randomly chosen from the set of agents A	The tasks enter the organisation, randomly, at any agent
Number of service instances per task	$\|si\|$	chosen from a uniform distribution between $1-20$	An upper limit of 20 is set so that the tasks do not become too bulky[1].
Number of tasks per simulation	$\|T\|$	100 and 200	Sufficient to load the organisation for meaningful results.
Time period of simulation		1000 time-steps	Sufficient amount of time for the obtaining meaningful results.
Start time of a task		chosen from a uniform distribution between $1-1000$	Since the tasks will be different with different service instances and dependencies, the incoming time of tasks is not very important.
Communication cost coefficient	C	0.25	For our present purposes, C is fixed at this value
Management load coefficient	M	0.50	So that management causes more load than message passing but less than actual service instance execution
Reorganisation cost coefficient	R	0	Currently, we set it to 0, so that the reorganisation process is not hindered by the cost of reorganising.
Maximum pattern size		$\frac{1}{3}\|si\|$	In the experiments where patterns are used, it is set to one-third of the maximum number of service instances in a task so that, in general, tasks need around three patterns.

2. The effectiveness of structural adaptation depends on the similarity between the tasks faced by the organisation. Structural reorganisation will be more effective if there is a high degree of similarity between the tasks.

The first hypothesis is reasonable because, in scenarios where the agents have very similar service sets, changing their relations will not result in much difference in terms of the connectivity between the different services, as the new relations will be connecting the same service sets as the old ones (as all agents have almost the same service sets). Similarly, if the tasks coming in to the organisation do not have any kind of similarity, then reorganisation by modifying the structure will not be very useful because there are no particular characteristics of the tasks that can be exploited through adaptation.

The controlled parameters of the environment which we predict will have a direct bearing on the $cost_{ORG}$ and $benefit_{ORG}$ are the ones described above— service probability (SP) and number of patterns (NoP). The rest of the parameters are either kept constant or randomly generated as specified in Table 5.1. We ran different experiments by varying one of the controlled parameters and keeping the other constant, to study the individual relationship between the parameters and the performance of the reorganisation procedures. Given this, we now list the different experiments that have been conducted:

1. Service probability (SP) is increased from 0 to 0.5 in steps of 0.1 with NoP set equal to 0 and 5 for organisations facing both 100 and 200 tasks.
2. Number of patterns (NoP) is varied from 1 to 10 with service probability fixed at 0 and 0.5 for organisations facing both 100 and 200 tasks.

The experiments are conducted by running simulations with the parameters set as described. For every simulation run, the set of services and agents is generated and then services are assigned to the agents on basis of the service probability SP. Next, the set of tasks is generated based on the number of patterns NoP and other parameters specified earlier. Then, an initial organisation structure is randomly generated from the set of agents and the simulation is run with this organisation executing the tasks. The average cost and benefit of that organisation for that simulation run is obtained by taking the average of the $cost_{ORG}$ and $benefit_{ORG}$ values that are generated at every time-step. The same simulation with the same task set and organisation structure (including the agents) is run for all the five reorganisation procedures (Smart, Random, Static, Central and Oracle). We conducted the experiments by running 1000 simulations runs (generating a new set of tasks and organisation each time) for every data point to achieve statistically significant results. Since the the 95% confidence intervals of the plotted means of Smart do not overlap with Oracle, Random or Static, we claim that our results are statistically significant.

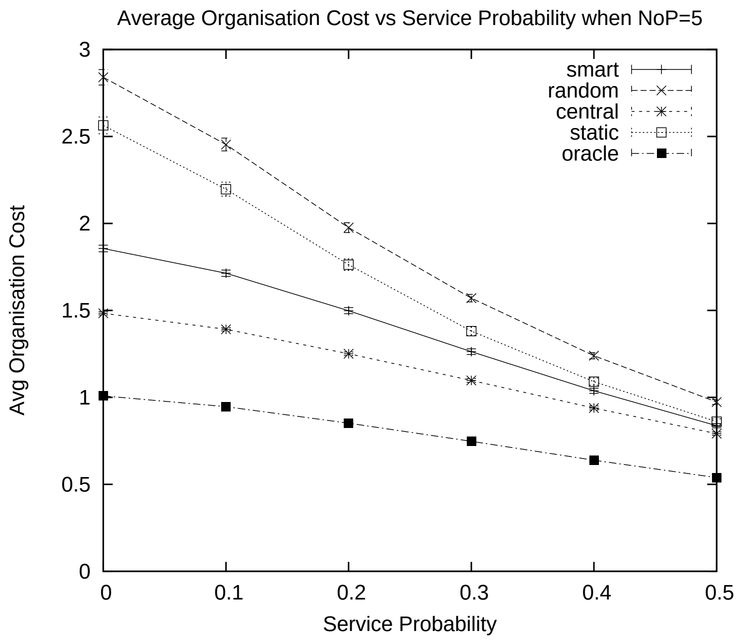
In the next section, we discuss and analyse the results obtained from the experiments.

5.2 Results

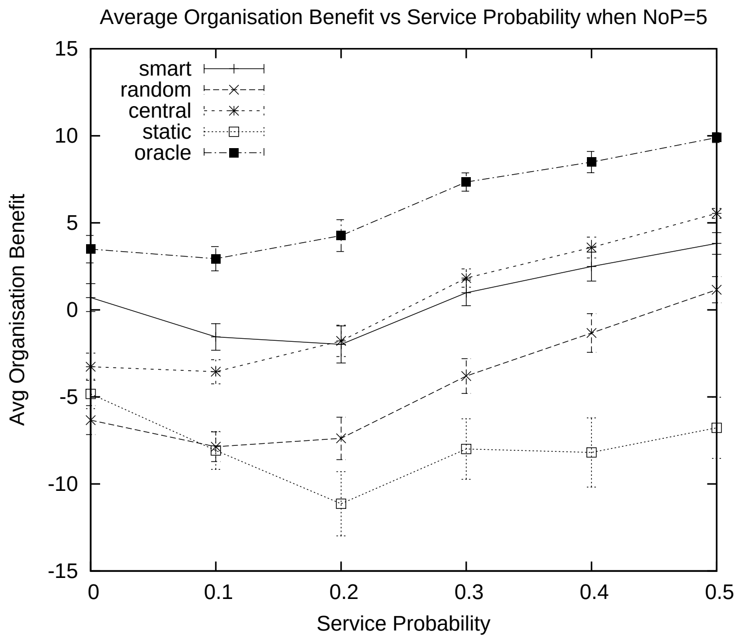
We present our results in the form of graphs plotting the cost and benefit values for all the five procedures. A better performance is reflected in a high benefit and a low cost result. As specified earlier, we conducted two sets of experiments— (i) organisations performing 100 tasks in a simulation run and (ii) organisations performing 200 tasks in a simulation run. We chose these two different sets to examine the possible change in performance of organisations that are moderately loaded and highly loaded.

We first discuss the experiments on moderately loaded organisations (100 tasks); Figs 5.1, 5.2, 5.3 and 5.4 display the results for organisations performing 100 tasks in a simulation run. In more detail, in Figs 5.1 and 5.2, the service probability increases along the x-axis from 0 (when all agents have a unique service set) to 0.5 (when every agent is capable of approximately half of the services). More specifically, Fig 5.1(a) and 5.1(b) display the cost and benefit when the tasks are highly similar (made up of 5 patterns only, $NoP = 5$), while in Fig 5.2(a) and 5.2(b) the tasks are completely dissimilar (not made of patterns, $NoP = 0$). For all the five reorganisation procedures, we see that the cost decreases as the similarity of the agents increases (see Figs 5.1(a) and 5.2(a)). This is because, as agents have increasingly similar service sets, most service instance delegations will require one assignment step only, as any agent will be capable of most services. However, considering both cost and benefit, we find that the Oracle outperforms all the other methods significantly, thus justifying our choice of an upper bound. Also, the Static and Random methods perform worse than the rest as neither do they adapt intelligently, nor do they use a centralised allocation mechanism. More importantly, we see that the performance of our method is very close to the Central strategy on both the occasions, thus showing that our method, which is decentralised in nature, succeeds in adapting the organisation structures that, in turn, leads to efficient allocation of tasks. Another interesting result is that the difference in the performance of all the methods is more apparent when the tasks are highly similar (when $NoP = 5$). This is because the organisation structure has more influence over the performance for similar tasks as a good structure, in such cases, leads to an efficient task allocation and vice versa.

Similar trends are also observed when the dissimilarity between tasks is increased along the x-axis (see Figs 5.3 and 5.4). The huge drop in benefit for all methods, seen in Fig 5.3(b) (the range has been limited to -40 for better readability, the complete graph can be seen in Fig 5.2) when the tasks are highly similar, occurs because the tasks are made up of only one pattern leading to a high demand for a few services. However, since the

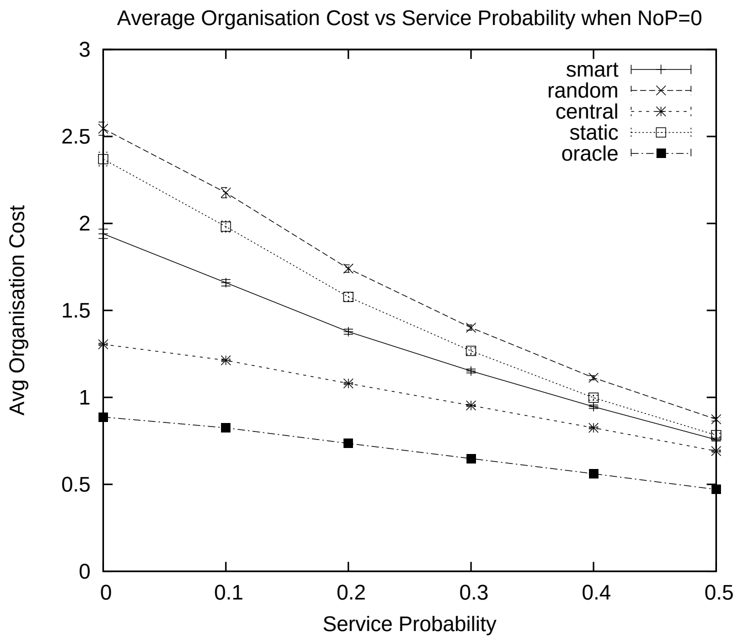


(a) Cost when NoP=5

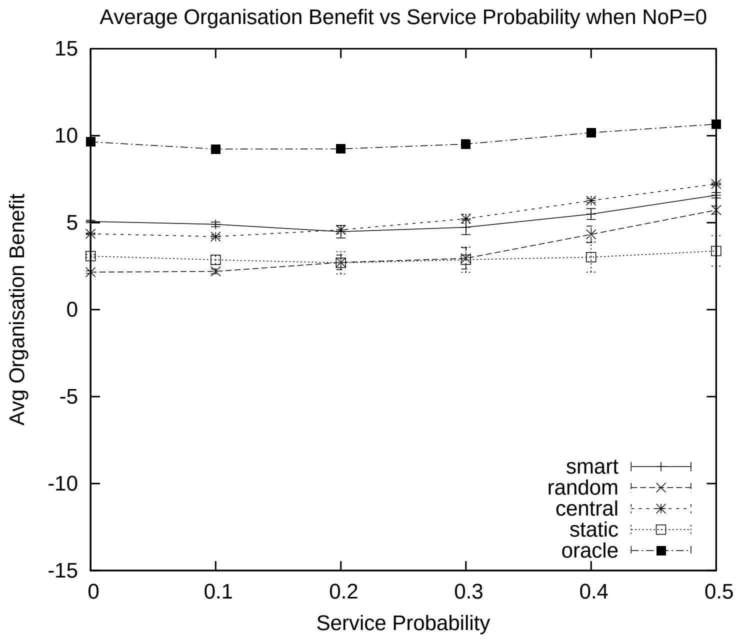


(b) Benefit when NoP=5

FIGURE 5.1: Average cost and benefit as similarity between agents increases for organisations facing highly similar 100 tasks

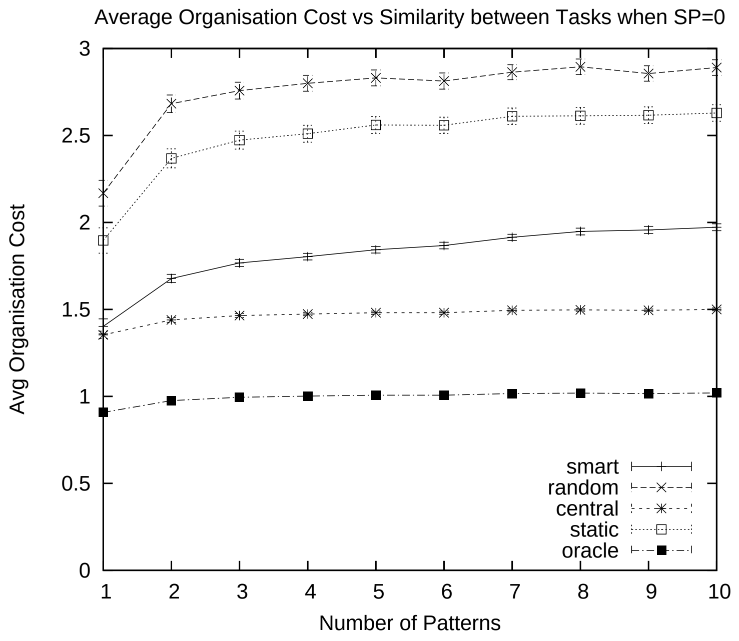


(a) Cost when NoP=0

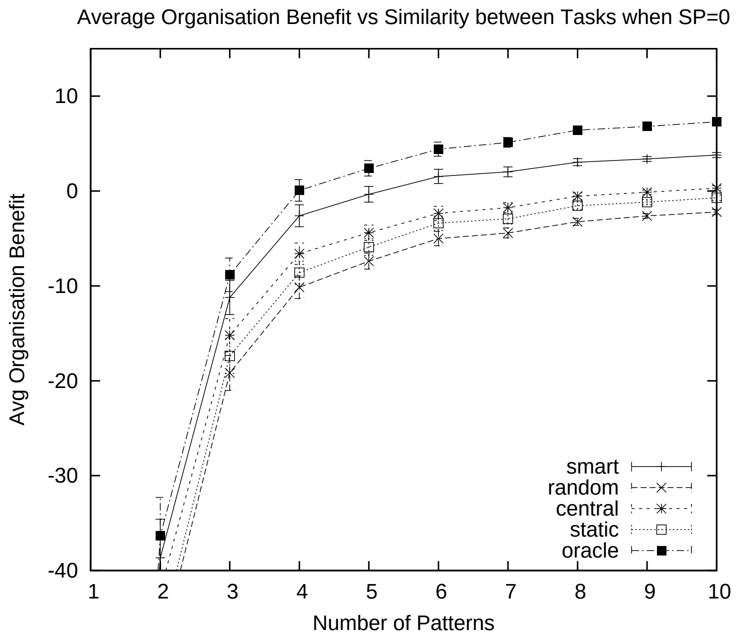


(b) Benefit when NoP=0

Figure 5.2: Average cost and benefit as similarity between agents increases for organisations facing dissimilar 100 tasks

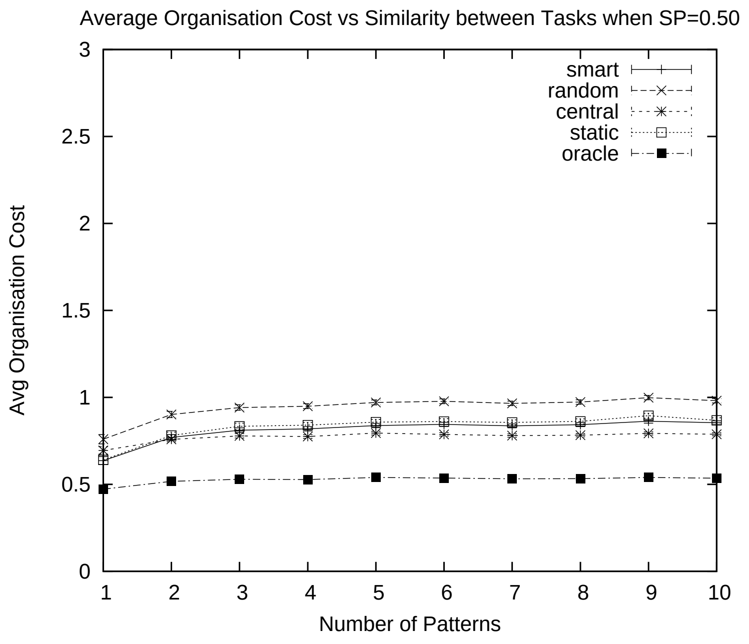


(a) Cost when SP=0

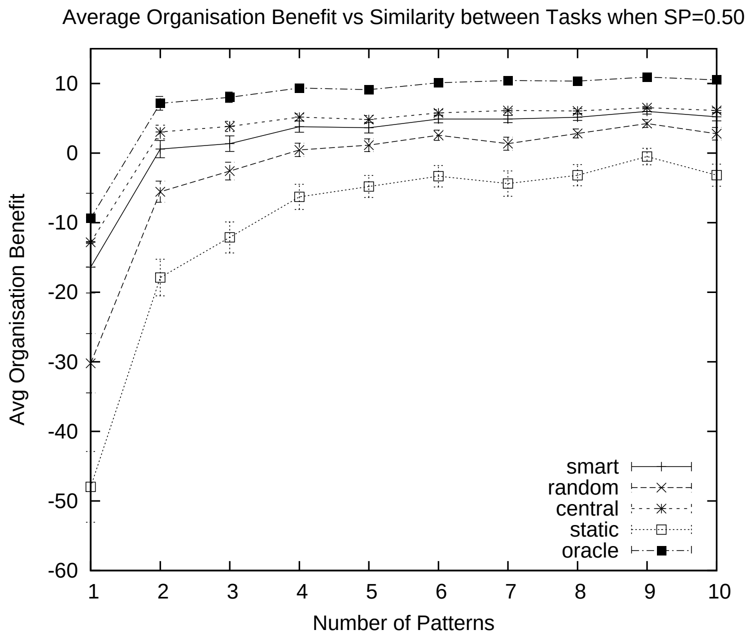


(b) Benefit when SP=0

FIGURE 5.3: Average cost and benefit as similarity between tasks decreases for organisations with unique agents facing 100 tasks



(a) Cost when SP=0.50



(b) Benefit when SP=0.50

Figure 5.4: Average cost and benefit as similarity between tasks decreases for organisations with highly similar agents facing 100 tasks

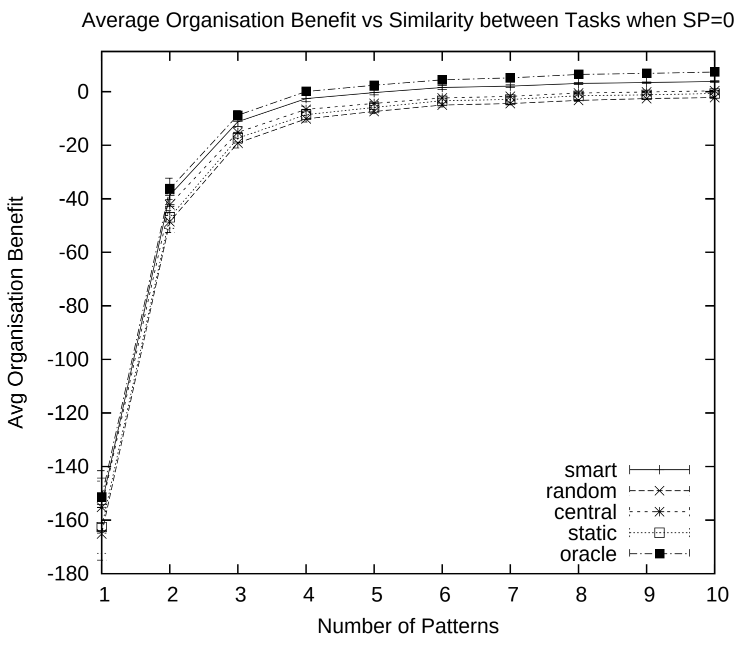
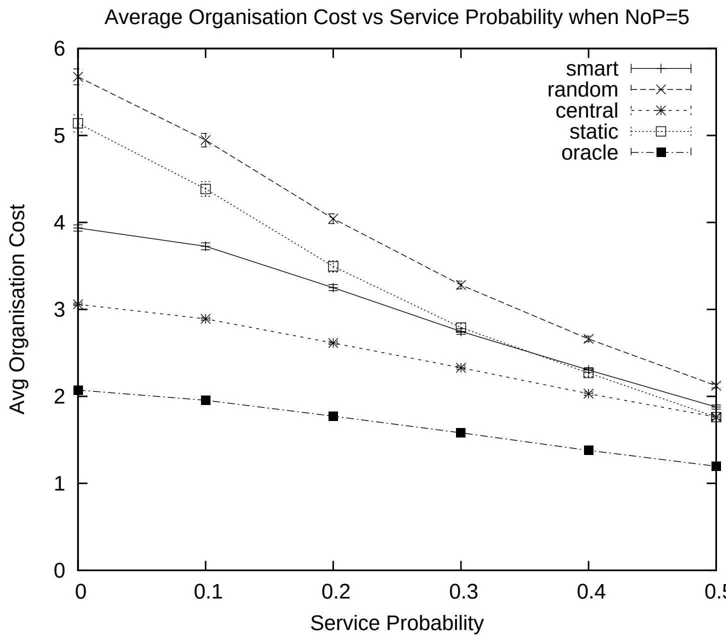


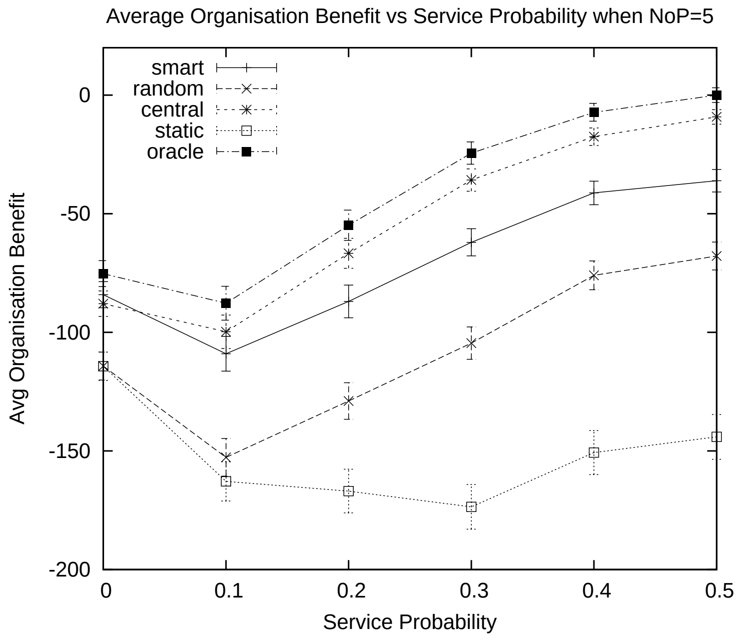
FIGURE 5.5: Fig 5.3(b) with the full y range

agents are unique, each of those services will be provided by one agent only, resulting in the overloading of those agents. This drop in benefit is less pronounced in Fig 5.4(b) because, here the agents have overlapping service sets and therefore a better distribution of load takes places. It is also interesting to note that our method performs significantly better than Central when the agents have unique service sets. This is because, the extra information possessed by the central agent about the load on agents is not really useful in these cases because only one agent is capable of any service and has to be allocated those corresponding service instances irrespective of the load on it. At the same time, one step delegations are possible in our method while the central method always requires a two-step delegation (with the central allocator as the intermediary), thus leading to a better performance of our method. Also, we see that there is a larger difference in costs across the five methods when the agents are unique (Fig 5.3(a)) than when the agents are similar (Fig 5.4(a)). This is because, as the cost is affected only by the length of the assignment chains, when the agents have similar sets, most service instances allocations are one step delegations irrespective of the structure (since most agents provide most services) and therefore the communication cost is low regardless of the reorganisation method.

With respect to our first hypothesis, in both Figs 5.1(b) and 5.2(b), we see that the difference in the benefits between Smart and Random is greater when $SP = 0$ than when $SP = 0.5$, thus demonstrating our hypothesis that structural adaptation is more effective when agents are unique. Moreover, our method performs better than Central when agents are unique and vice versa further supporting our hypothesis. Similarly, in

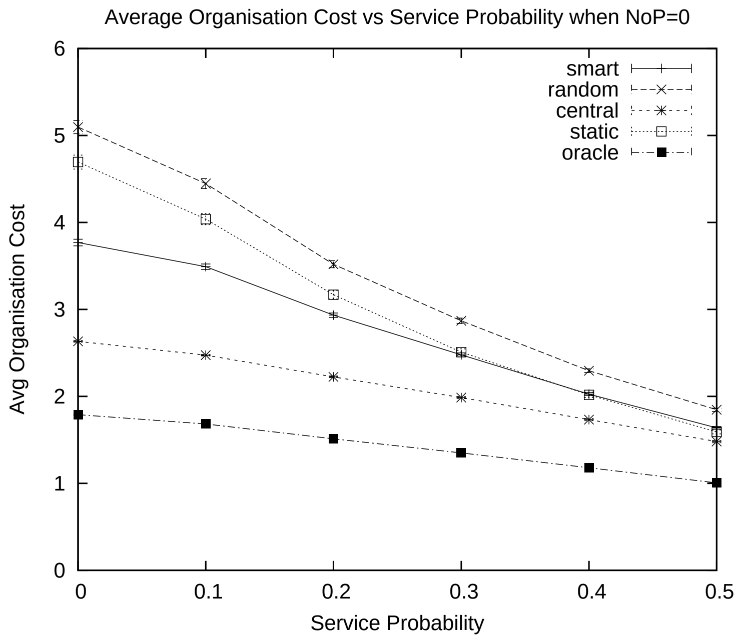


(a) Cost when NoP=5

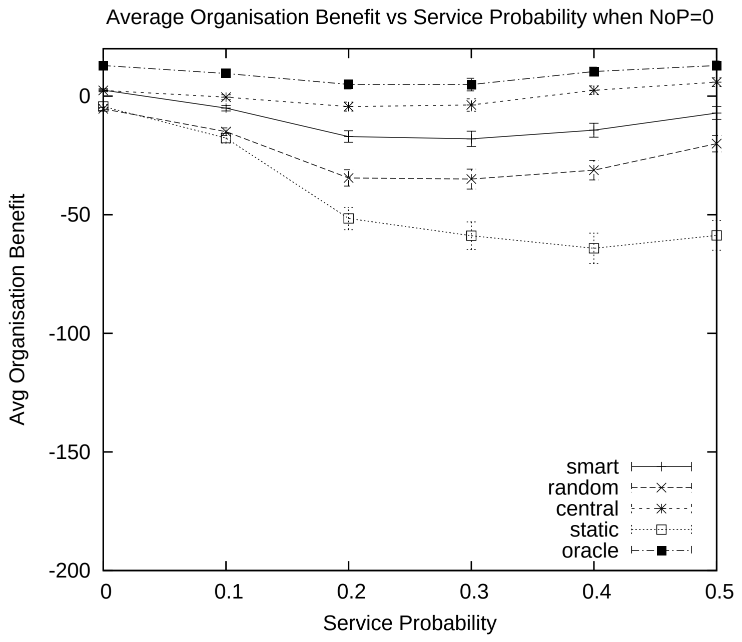


(b) Benefit when NoP=5

FIGURE 5.6: Average cost and benefit as similarity between agents increases for organisations facing highly similar 200 tasks

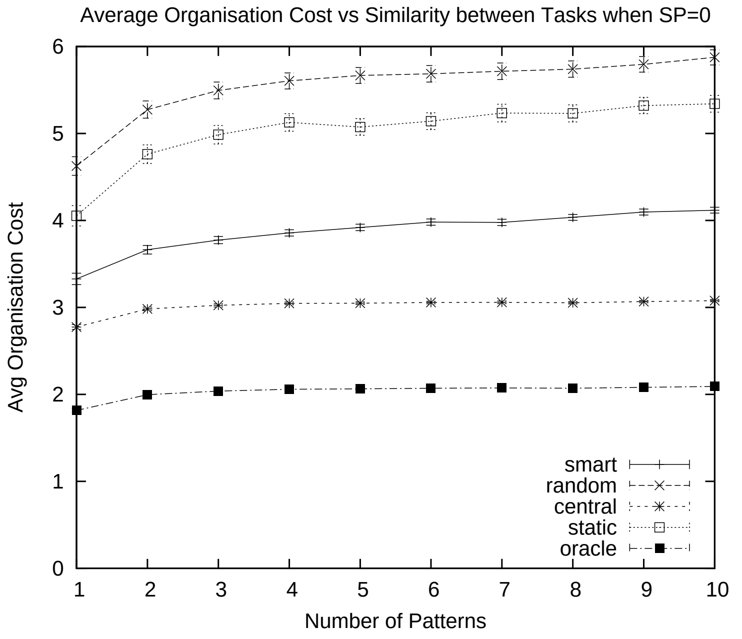


(a) Cost when NoP=0

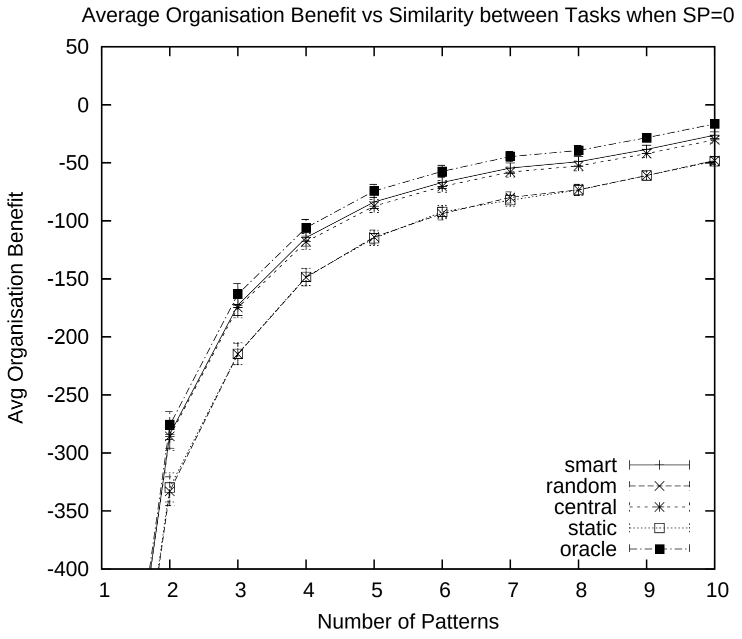


(b) Benefit when NoP=0

FIGURE 5.7: Average cost and benefit as similarity between agents increases for organisations facing dissimilar 200 tasks

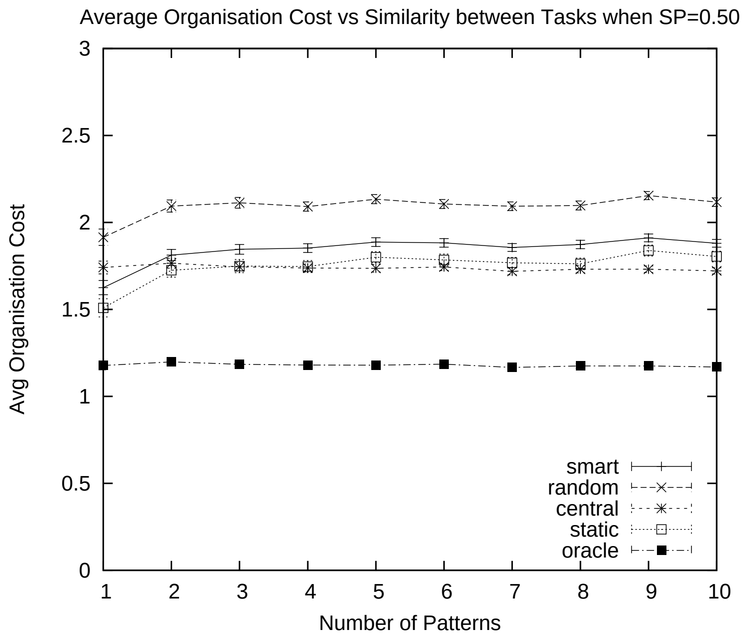


(a) Cost when SP=0

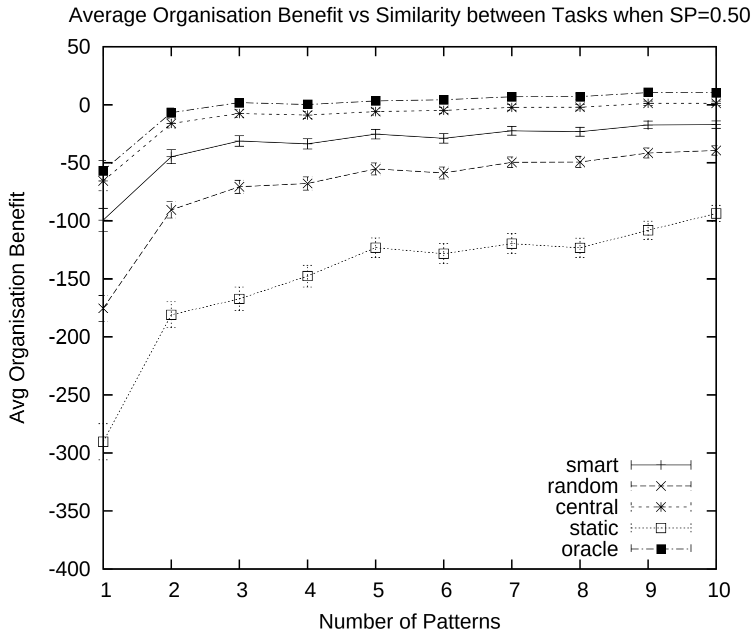


(b) Benefit when SP=0

Figure 5.8: Average cost and benefit as similarity between tasks decreases for organisations with unique agents facing 200 tasks



(a) Cost when SP=0.50



(b) Benefit when SP=0.50

FIGURE 5.9: Average cost and benefit as similarity between tasks decreases for organisations with highly similar agents facing 200 tasks

Figs 5.3(b) and 5.4(b), we find that Smart and Random come closer in performance when the number of patterns is increased along the x-axis. Therefore, this demonstrates our second hypothesis that structural adaptation is more useful when the tasks are similar (fewer number of patterns) than otherwise.

As stated earlier, we have also experimented by increasing the workload on the organisation by supplying it with 200 tasks over the same 1000 time-steps. In this case, we find that the resultant trends are similar to those in organisations faced with 100 tasks. That is, the earlier comparison between the different reorganisation procedures is still valid when the organisations are overloaded. If any, the difference in the performance between the different methods is more pronounced in this case, as the advantage of an efficient structure over an inefficient one is highlighted when more and more tasks are performed. The only difference between the two sets of experiments is that our method does not perform significantly better than the Central when the agents have unique service sets (Fig 5.8(b)) unlike the earlier case (Fig 5.3(b)). This is because the overloaded agents (due to the high number of tasks) in our method will keep searching, unsuccessfully, for subordinates to reassign their service instances which then leads to more management load. However, in such cases, they are not able to assign the service instances to anyone else because the agents are unique and only one of them is capable of a particular service. On the other hand, in the Central method, the central allocator will make the delegations and the agents do not use up their capacity searching for the other agents for reassignment.

In summary, we find that our reorganisation method always performs better than a random reorganisation or a non-reorganising approach. Furthermore, our method performs similarly to the approach based on an external omniscient central agent performing the allocations.

5.3 Summary

In this chapter, we presented the experimental setup used for testing our structural adaptation mechanism and comparing it against organisations without reorganisation, those employing a random reorganisation strategy, those employing an external central omniscient allocator and those having omniscient agents which do no incur any management load for allocations. From our experiments, we found that our mechanism always performed better than the non-reorganising and randomly reorganising approaches. Furthermore, its performance was comparable to the organisations with the central external agent. Therefore, it is evident that our method is successful in improving the efficiency of the organisation and brings its performance closer to that of a centralised infinitely resourceful allocator than a randomly adapting or a non-adapting organisation. Therefore, we claim that our adaptation mechanism satisfies the requirement of developing a decentralised agent-based structural adaptation method for improving the efficiency of problem-solving organisations. However, our method currently functions only in a constrained environment and this provides several directions for future work.

In the next chapter, the long term goals of this research are detailed along with a timeline for the final thesis.

Chapter 6

Conclusions and Future Work

This report describes our approach to the problem of developing decentralised structural adaptation mechanisms for problem-solving agent organisations based on the paradigm of self organisation. We started by examining the existing models for depicting agent organisations (Section 2.1) and also studied the various self-organisation approaches being applied in multi-agent systems (Section 2.2). More specifically, we focused on the current adaptation methods for agent organisations and highlighted the deficiencies of the present approaches, mainly that none of them provide a robust decentralised method for purely modifying the organisation structure in order to improve the performance of the organisation. We went on to define a simple organisation model (Chapter 3) to serve as a framework for developing structural adaptation methods. The model comprised a task environment with a continuous stream of tasks which are made up of service instances containing dependencies among themselves and an organisation of agents that are capable of providing the required services to accomplish the tasks. The agents are related to each other on the basis of the organisation structure. The agent relations, which are defined by the organisation structure, also determine the possible interactions between the agents. Then, we presented our structural adaptation method (Section 4.2) using a decision theoretic approach in the form of actions and their expected utilities that will be used by all the agents in the organisation to modify their relations with other agents. This reorganisation mechanism is based on the history of task allocations by the agents and results in the structural modification of the organisation. Finally, we conducted sets of experiments (Section 5.1) to investigate the effectiveness of this method by comparing it against four other reorganisation procedures. Analysing the results (Section 5.2), we found that our method performed significantly better than a non-reorganising approach or a randomly reorganising approach. Moreover, the performance of the organisations employing our method was comparable to those using an external omniscient agent for allocation of tasks. Therefore, our decentralised adaptation method was successful in improving the efficiency of problem-solving organisations by appropriately modifying the organisation structure.

Looking back at the research objectives (stated in Section 1.1) of this work, we see that we have made good progress towards addressing the first and second requirements by creating a simple model of a problem-solving agent organisation and an evaluation mechanism that calculates efficiency via the cost and benefit of the organisation. With respect to the third requirement, we have developed a completely decentralised structural adaptation method that improves the efficiency of the organisation. Finally, we have also addressed the last requirement by carrying out an analysis of the performance of our method under different experimental conditions. As a result of working towards these requirements, we have succeeded in advancing the state of the art in the domain of problem-solving agent organisations by developing a novel decentralised structural adaptation method for improving the efficiency of such organisations in a constrained environment. Nevertheless, we need to further extend our work so that we make a fuller contribution by developing a more comprehensive and sophisticated approach.

Next, we analyse the shortcomings of our structural adaptation method to help uncover avenues for future work to achieve our objectives.

6.1 Future Work

Since, we have attempted to satisfy the research objectives of this report by making several assumptions and placing some constraints on the model, it provides us with scope for more work in the future. By critically analysing our adaptation mechanism, in its current form, we can identify a few potential shortcomings:

- The method is primarily designed with the assumption that the tasks coming in the future are somewhat similar to the tasks in the past. In the worst case scenario, if the tasks have contrasting dependency graphs, then reorganisation on the basis of past task assignments may not be very useful since the tasks in the future may require completely different assignment patterns. Moreover, the method does not utilise any information about the task environment (if provided) to be proactive and make anticipatory changes to the structure.
- The method requires that every pair of agents evaluate their relation at every time-step making it an inefficient process as most of the relations will remain unchanged over reasonably moderate periods of time.

- The adaptation is carried out by pairs of agents modifying their inter-relations. However, the method fails to directly look at possible reorganisation actions involving more than two agents, and therefore, more than one relation at the same time.

The first drawback is mainly an effect of the task environment which does not provide any preliminary information to the agents. We do not intend to directly tackle this issue but will persist with developing adaptation methods that are not specialised for any particular type of task environments. Taking into account the other two drawbacks and some of the constraints earlier placed on the model (see Section 4.2.1), we divide the future work into three stages as shown in Fig 6.1.

6.1.1 Task 1: Efficient Structural Adaptation

The first stage involves coming up with an efficient algorithm for structural adaptation (Item 3 of Task 1). As discussed above, the current method is $O(n^2)$, that is, quadratic in the number of agents as every agents needs to look at all its relations at every time-step. Therefore, we should develop a mechanism using which the agents will be able to prioritise their relations. Using this priority ranking, an agent will need to re-evaluate only a small subset of its relations at a time-step, thereby making the adaptation process more computationally efficient. At the same time, we also want to apply a different mechanism for the evaluation of the organisations to more rigorously verify the performance of our method (Item 1 of Task 1). Unlike the current efficiency evaluation which is based on the benefits and costs garnered by the individual agents, the new evaluation mechanism will be completely separated from the agents and be purely based on the completion time of the tasks. Furthermore, we also intend to relax the constraints placed on the environment and conduct experiments by varying the communication cost and management load coefficients to test how our reorganisation method copes with these changes (Item 2 of Task 1).

6.1.2 Task 2: Structural Adaptation in Dynamic Organisations

The other primary constraints currently placed on the framework, for developing our adaptation mechanism, are that the agents are invariant and the organisation is *closed* (no new agents enter or old agents leave). The second stage of the future work involves relaxing these two constraints. We will first try to enhance our method so that the structure of the organisation is able to adapt when the service sets of the agents are varying in the course of a simulation run (Item 1 of Task 2). Following that, we aim to extend it to cope with dynamic organisations which have agents entering and leaving the system (Item 2 of Task 2).

6.1.3 Task 3: Multi-step Structural Adaptation

The final stage of future work will address the last drawback listed above. We intend to develop a multi-step adaptation approach which works by involving three or more agents at the same time and therefore resulting in multiple steps of restructuring. This approach will require some kind of planning on the part of the agents so that they are able to look ahead and carry out a sequence of reorganisation actions. We will first develop such an approach for the constrained model of a closed organisation with invariant agents (Item 1 of Task 3) and later extend it to dynamic organisations (Item 2 of Task 3).

In conclusion, our future work aims to developed a sophisticated and comprehensive decentralised agent-based structural adaptation mechanism that aids in the improvement of the efficiency of problem-solving agent organisations in dynamic environments.

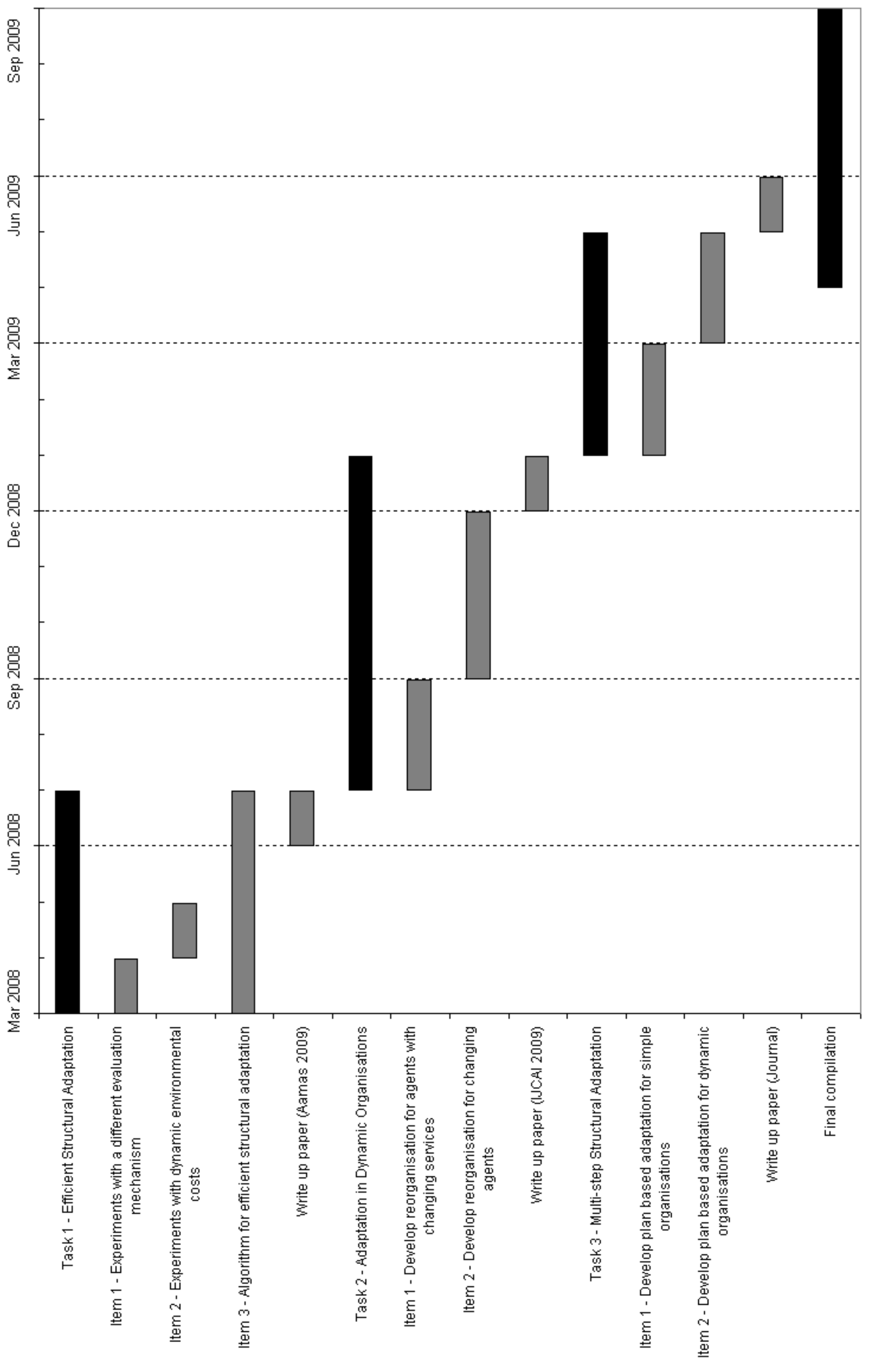


FIGURE 6.1: Time line for future work

Appendix A

Glossary

1. **Service:** We define a service as a specialised atomic action that can be executed by an agent. S is defined as the set of services provided by an organisation.

2. **Service instance:** An instance of a specific service has three parameters. It specifies the type of service required, the computational rate required and the total number of time-steps for which it is required. Hence, we define a service instance to be $si_i = \langle s_i, p_i, n_i \rangle$ where $s_i \in S$, $p_i \in \mathbb{N}$ denotes the amount of computation required per time-step and $n_i \in \mathbb{N}$ denotes the total time-steps it is required. SI denotes the set of all service instances of all tasks.

3. **Dependency:** A service instance is said to be dependent on another service instance when the output of the latter is required by the former. Hence the execution of the former cannot begin until the execution of latter is finished. It is represented formally in the dependency links graph.

4. **Dependency links:** The set of dependency links D_i contains links between the various si_j of the task. These links are directed arcs between any two service instances depicting the dependency of the source on the destination. An element d_j of D_i is of the form $d_j = \langle si_x, si_y \rangle$ where si_x and si_y are the origin and destination of the link.

5. **Task:** A task is composed of a number of service instances with a precedence order. It is defined as a tuple containing a set of service instances and a set of dependency links between the service instances. $t_i = \langle \{si_j \in SI, j \in \mathbb{N}\}, D_i \rangle$ (Eqn 3.1).

6. **Agent:** Agents are independent computational entities capable of providing services. Every agent has two parameters, a set of services that it can provide and a computational capacity. Let A be the set of agents. Every element of A is of the

form $a_x = \langle S_x, L_x \rangle$ where $S_x \in S$ denotes the services set of the agent and $L_x \in \mathbb{N}$ represents the computational capacity (Eqn 3.2).

7. **Computational capacity:** This is defined as the maximum computational load that an agent can undertake in a single time-step. It is denoted by L_x for agent a_x.

8. **Acquaintance relation:** All agents whose presence is known to an agent constitute the acquaintances of that agent. Acquaintance relations are formally represented by the *Acqt* link in the organisation graph.

9. **Superior-subordinate relation:** An agent is the superior of another agent if the agent has a superior-subordinate relation with the other agent. The superior has a *Supr* link to its subordinate in the organisation graph.

10. **Peer relation:** Two agents are considered peers if they have a peer relation between them. Peer relations are formally represented by the *Peer* links in the organisation graph.

11. **Accumulated service set:** The accumulated service set of an agent is the union of its own service set and the accumulated service sets of its subordinates, recursively. For an agent a_x, it is denoted by AS_x.

12. **Organisation graph:** The relationships between the agents in an organisation are represented in a organisation graph G. Every link r_i that belongs to G is of the form $r_i = \langle a_x, a_y, type_i \rangle$ where a_x and a_y are agents that the link originates and terminates respectively and $type_i$ denotes the type of link and it can take values $\{Acqt, Supr, Peer\}$ representing the three types of relations possible (Eqn 3.4).

13. **Organisation:** The Organisation consists of the set of agents and a set of organisational links. Therefore, it can be represented by a 2-tuple of the form $ORG = \langle A, G \rangle$ where A is the set of agents and G is the set of directed links between the agents (Eqn 3.3).

14. **Communication cost coefficient:** This denotes the cost in terms of the amount of resource required by the network to transmit one message between two agents. It is denoted by C

15. **Management load coefficient:** This denotes the amount of computation that is spent by an agent for evaluating whether a particular agent could be assigned a particular service instance. It is denoted by M

16. **Cost of an agent:** It denotes the amount of resource used up by the system for transmitting the messages of an agent. It is calculated for every time-step. The cost of an agent a_x is $cost_x = C.c_x$ where c_x is the number of messages sent by the agent a_x (Eqn 3.5).

17. **Cost of the organisation:** The cost of the organisation is the total amount of computational units utilised by the network for the transmission of the messages between the agents. It is calculated for every time-step:

$$cost_{ORG} = C.\sum_{x=0}^{|A|} c_x$$

(Eqn 3.6)

18. **Benefit of an agent:** It denotes the reward gained by the organisation due to an agent. It is calculated for every time-step. The benefit of an agent at a time-step is equal to the total amount of computation of the service instances being executed by that agent subtracted by the amount of computation required by the service instances that are waiting to be executed or allocated by the agent. Therefore, the benefit of agent a_x:

$$benefit_x = \sum_{i=0}^{|T_{x_E}|} e_{ix} - \sum_{i=0}^{|T_{x_W}|} e_{ix}$$

T_{x_E} is the set of tasks being executed by the agent and T_{x_W} is the set of tasks waiting for allocation or execution by the agent. e_{ix} in each case represents the sum of the computational rates of the service instances of the task t_i that are being executed or waiting at the agent a_x (Eqn 3.8).

19. **Benefit of the organisation:** It denotes the overall reward gained by the organisation. It is calculated as the sum of the rewards obtained by the individual agents:

$$benefit_{ORG} = \sum_{x=0}^{|A|} benefit_x$$

(Eqn 3.9).

20. **Efficiency of the organisation:** It denotes the overall performance of the organisation. It is denoted as $efficiency_{ORG} = benefit_{ORG} - cost_{ORG}$ (Eqn 3.10)

21. **Reorganisation cost coefficient:** It denotes the cost of forming or dissolving a relation between two agents. It is denoted by R.

22. $form\_subr_{x,y}$: It denotes the action of forming a superior-subordinate relation between agents a_x and a_y, with a_x as the superior.

23. $rem\_subr_{x,y}$**:** It denotes the action of dissolving a superior-subordinate relation between agents a_x and a_y where a_x was the superior.

24. $form\_peer_{x,y}$: It denotes the action of forming a peer relation between agents a_x and a_y.

25. $rem\_peer_{x,y}$**:** It denotes the action of dissolving a peer relation between agents a_x and a_y.

26. **Assignment:** By assignment of a service instance to an agent, we mean that the agent has been allocated that service instance. It must accomplish that service instance by either executing it itself or by assigning it again to some other agent. Formally, when a service instance si_i is assigned to agent a_x, then a_x is considered an *assigned agent* for si_i.

27. **Delegation:** By delegation of a service instance to an agent, we refer to the fact that the agent is executing the particular service instance by itself (without reassigning it to someone else). Formally, when an agent a_x executes a service instance si_i, it is considered as the *delegated agent* for si_i. Thus, there could be several assigned agents for a particular service instance, but only one delegated agent.

28. **Value function:** It denotes the function that gives the values of the expected utilities of performing a reorganisation action between agents a_x and a_y. It depends on the estimated change in the load and costs of the two agents in question and the other agents involved in the delegations between these two agents. It is calculated as:

$$V = \delta load_x + \delta load_y + \delta load_{OA} + \delta cost_{comm} + cost_{reorg} \tag{A.1}$$

(Eqn 4.1)

29. $Asg_{x,y}$**:** The number of service instances assigned by an agent a_x to a_y.

30. $Del_{x,y}$**:** The number of service instances delegated by an agent a_x to a_y.

31. n^{tot}**:** The total number of time-steps that have elapsed since the beginning of the run.

32. $n^{subr}_{x,y}$**:** The number of time-steps that a_x and a_y had a superior-subordinate relation (that is, time-steps that $\langle a_x, a_y, Supr\rangle \in G$) . Likewise, $n^{peer}_{x,y}$ denotes the amount of time that a_x and a_y had a peer relation.

33. $filled_x(n)$: The number of time-steps out of the total time denoted by n that a_x had waiting tasks ($T_{xW} \neq \phi$; capacity being completely filled by load). The variable n can represent the total time passed (n^{tot}) or the time duration that a_y was its peer ($n^{peer}_{x,y}$) and so on.

34. $Asg_{x,subr}$**:** The number of service instances that have been assigned by a_x to any of its subordinates. Likewise, $Asg_{x,peer}$ and $Asg_{x,supr}$.

35. $Asg_{x,tot}$**:** The total number of service instances that have been assigned by a_x to other agents. Therefore, $Asg_{x,tot} = Asg_{x,subr} + Asg_{x,peer} + Asg_{x,supr}$.

36. $Asg^{LOAD}_{x,y}$: The management load added onto a_y because of assignments from a_x (the count of these assignments is denoted by $Asg_{x,y}$ as stated above).

37. $IA_{x,y}$**:** The total number of times, other agents (intermediate agents) were involved in the delegations of service instances by a_x to a_y. Therefore,

$$IA_{x,y} = \sum_{i}^{Del_{x,y}} count^i_{OA}$$

where $count^i_{OA}$ is the number of other agents involved in the delegation of si_i from a_x to a_y.

38. $IA^{COST}_{x,y}$**:** The communication cost due to the delegations from a_x to a_y. For every agent in $IA_{x,y}$, a cost of $2C$ is added because a message is once sent forward and once back. Therefore, $IA^{COST}_{x,y} = IA_{x,y} * 2 * C$.

39. $IA^{LOAD}_{x,y}$**:** The overall management load put on all the intermediate agents involved in the delegations from a_x to a_y (that is, $Del_{x,y}$). The load values are reported back to a_x along with the assignment information (the assignment message). If an intermediate agent has available capacity (no waiting tasks), it will report a 0 load value for that delegation. Otherwise, the agent will report the actual management load that was put on it due to that service instance assignment.

Bibliography

Bernon, C., Chevrier, V., Hilaire, V., and Marrow, P. (2006). Applications of self-organising multi-agents systems: An initial framework of comparison . *Informatica*, 30(1):73–82.

Beverly, R. and Afergan, M. (2007). Machine learning for efficient neighbor selection in unstructured p2p network. In *USENIX Tackling Computer Systems Problems with Machine Learning Techniques (SysML'07)*, Cambridge, MA, USA.

Bollen, J. and Heylighen, F. (1996). Algorithms for the self-organisation of distributed, multi-user networks. possible application to the future world wide web. *Cybernetics and Systems '96, Austrian Society of Cybernetics*, pages 911–916.

Bongaerts, L. (1998). *Integration of Scheduling and Control in Holonic Manufacturing Systems*. PhD thesis, PMA/K.U. Leuven.

Bou, E., Lopez-Sanchez, M., and Rodriguez-Aguilar, J. A. (2006a). Norm adaptation of autonomic electronic institutions with multiple goals. *International Transactions on Systems Science and Applications*, 1(3):227–238.

Bou, E., Lopez-Sanchez, M., and Rodriguez-Aguilar, J. A. (2006b). Self-configuration in autonomic electronic institutions. In *Coordination, Organization, Institutions and Norms in Agent Systems Workshop at ECAI Conference*, pages 1–9, Trentino, Italy.

Bourjot, C., Chevrier, V., and Thomas, V. (2003). A new swarm mechanism based on social spiders colonies: From web weaving to region detection. *Web Intelligence and Agent Systems*, 1(1):47–64.

Capera, D., George, J.-P., Gleizes, M.-P., and Glize, P. (2003a). The AMAS theory for complex problem solving based on self-organizing cooperative agents. In *Proceedings of the 12th International Workshop on Enabling Technologies (WETICE '03)*, page 383, Washington, DC, USA. IEEE Computer Society.

Capera, D., Gleizes, M. P., and Glize, P. (2003b). Self-organizing agents for mechanical design. In *Engineering Self-Organising Systems*, volume 2977 of *LNCS*, pages 169–185. Springer.

Carley, K. M. and Gasser, L. (1999). Computational organization theory. In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 299–330. MIT Press, Cambridge, MA, USA.

Cohen, P. R. and Levesque, H. J. (1991). Confirmations and joint action. In *Proceedings of the twelfth International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 951–959, Sydney, Australia. Morgan Kaufmann Inc.

De Wolf, T. and Holvoet, T. (2003). Towards autonomic computing: agent-based modelling, dynamical systems analysis, and decentralised control. In *Proceedings of the First International Workshop on Autonomic Computing Principles and Architectures*, pages 10–20, Banff, Canada.

Di Marzo Serugendo, G., Gleizes, M.-P., and Karageorgos, A. (2005a). Self-organisation in multi-agent systems. Rapport de recherche IRIT/2005-18-R, IRIT, Universite Paul Sabatier, Toulouse.

Di Marzo Serugendo, G., Gleizes, M.-P., and Karageorgos, A. (2005b). Self-organization in multi-agent systems. *The Knowledge Engineering Review*, 20(2):165–189.

Di Marzo Serugendo, G., Gleizes, M.-P., and Karageorgos, A. (2006). Self-organisation and emergence in multi-agent systems: An overview . *Informatica*, 30(1):45–54.

Dignum, V. (2003). *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, Proefschrift Universiteit Utrecht.

Dignum, V. and Dignum, F. (2005). Structures for agent organizations. In *International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, Waltham, USA. IEEE Computer Society.

Dignum, V., Dignum, F., and Sonenberg, L. (2004). Towards dynamic reorganization of agent societies. In *Proceedings of the Workshop on Coordination in Emergent Agent Societies (WCES) at ECAI'04*, pages 22–27, Valencia, Spain.

Dowling, J., Cunningham, R., Curran, E., and Cahill, V. (2006). Building autonomic systems using collaborative reinforcement learning. *The Knowledge Engineering Review*, 21(3):231–238.

Ferber, J. and Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the 3rd International Conference on Multi Agent Systems (ICMAS '98)*, pages 128–135, Washington, DC, USA. IEEE Computer Society.

Ferber, J., Gutknecht, O., and Michel, F. (2003). From agents to organizations: An organizational view of multiagent systems. In *Proceedings of the Fourth International Workshop on Agent Oriented Software Engineering (AOSE03)*, page 214230, Melbourne, Australia. Springer Verlag.

Fischer, K. (2005). Self-organisation in holonic multiagent systems. In *Mechanizing Mathematical Reasoning*, volume 2605, pages 543–563. Springer.

Fox, M. S. (1988). *An organizational view of distributed systems*, pages 140–150. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Galbraith, J. R. (1977). *Organization Design*. Addison-Wesley, Reading, MA.

Gasser, L., Braganza, C., and Herman, N. (1988). Implementing distributed AI systems using MACE. *Distributed Artificial Intelligence*, pages 445–450.

Gasser, L. and Ishida, T. (1991). A dynamic organizational architecture for adaptive problem solving. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI '91)*, pages 185–190, Anaheim, CA.

Gershenson, C. (2007). *Design and Control of Self-organizing Systems*. PhD thesis, Vrije Universiteit Brussel.

Grossi, D., Dignum, F., Dignum, V., Dastani, M., and Royakkers, L. (2006). Structural evaluation of agent organizations. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems (AAMAS '06)*, pages 1110–1112, New York, NY, USA. ACM Press.

Hannoun, M., Boissier, O., Sichman, J. S., and Sayettat, C. (2000). Moise: An organizational model for multi-agent systems. In *Proceedings of the International Joint Conference, 7th Ibero-American Conference on AI (IBERAMIA-SBIA '00)*, pages 156–165, London, UK. Springer-Verlag.

Hassas, S., Di Marzo Serugendo, G., Karageorgos, A., and Castelfranchi, C. (2006). Self-organising mechanisms from social and business/economics approaches. *Informatica*, 30(1):63–71.

Holland, J. H. (1998). *Emergence: from chaos to order*. Addison-Wesley, Reading, MA, USA.

Hoogendoorn, M. (2007). Adaptation of organizational models for multi-agent systems based on max flow networks. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI '07)*, pages 1321–1326, Hyderabad, India. AAAI Press.

Hoogendoorn, M., Jonker, C. M., and Treur, J. (2007). Redesign of organizations as a basis for organizational change. In *Coordination, Organizations, Institutions, and Norms in Agent Systems II (COIN'06 workshops)*, volume 4386 of *LNAI*, pages 46–62. Springer.

Horling, B., Benyo, B., and Lesser, V. (2001). Using self-diagnosis to adapt organizational structures. In *Proceedings of the fifth international conference on Autonomous agents (AGENTS '01)*, pages 529–536, New York, NY, USA. ACM Press.

Horling, B. and Lesser, V. (2005). A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(4):281–316.

Horling, B. and Lesser, V. (2008). Using quantitative models to search for appropriate organizational designs. *Autonomous Agents and Multi-Agent Systems*, 16(2):95–149.

Horn, P. (2001). Autonomic Computing: IBM's Perspective on the State of Information Technology.

Hubner, J. F., Sichman, J. S., and Boissier, O. (2004). Using the MOISE+ for a cooperative framework of MAS reorganisation. In *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence (SBIA'04)*, volume 3171, pages 506–515, Berlin. Springer.

Ishida, T., Gasser, L., and Yokoo, M. (1992). Organization self-design of distributed production systems. *IEEE Transactions on Knowledge and Data Engineering*, 4(2):123–134.

Itao, T., Nakamura, T., Matsuo, M., Suda, T., and Aoyama, T. (2002). Service emergence based on relationship among self-organizing entities. In *Proceedings of the 2002 Symposium on Applications and the Internet (SAINT '02)*, pages 194–203, Washington, DC, USA. IEEE Computer Society.

Jelasity, M. and Babaoglu, O. (2005). T-man: Gossip-based overlay topology management. In *Proceedings of Engineering Self-Organising Applications (ESOA'05)*, Utrecht, The Netherlands. Springer.

Jin, Y. and Levitt, R. E. (1996). The virtual design team: A computational model of project organizations. *Computational & Mathematical Organization Theory*, 2:171–196(26).

Kamboj, S. and Decker, K. S. (2006). Organizational self-design in semi-dynamic environments. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06)*, pages 335–337, New York, NY, USA. ACM Press.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *IEEE Computer*, 36(1):41–50.

Klein, F. and Tichy, M. (2006). Building reliable systems based on self-organizing multi-agent systems. In *Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems (SELMAS '06)*, pages 51–58, New York, NY, USA. ACM Press.

Knabe, T., Schillo, M., and Fischer, K. (2003). Inter-organizational networks as patterns for self-organizing multiagent systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems (AAMAS'03)*, pages 1036–1037, New York, NY, USA. ACM.

Krackhardt, D. and Carley, K. M. (1998). A pcans model of structure in organizations. *Proceedings of the 1998 International Symposium on Command and Control Research and Technology*, pages 113–119.

Lematre, C. and Excelente, C. B. (1998). Multi-agent organization approach. In *Proceedings of second Ibero-American Workshop on DAI and MAS*, Toledo, Spain.

Mainsah, E. (2002). Autonomic computing: the next era of computing. *Electronics & Communication Engineering Journal*, 14(1):2–3.

Mamei, M., Vasirani, M., and Zambonelli, F. (2004). Self-organizing spatial shapes in mobile particles: The tota approach. In *Engineering Self-Organising Systems, Methodologies and Applications (ESOA 04)*, pages 138–153, New York, USA.

Mano, J.-P., Bourjot, C., Lopardo, G., and Glize, P. (2006). Bio-inspired mechanisms for artificial self-organised systems. *Informatica*, 30(1):55–62.

McGinty, L. and Smyth, B. (2002). Shared experiences in personalized route planning. In *Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference*, pages 111–115. AAAI Press.

Mills, K. L. (2007). A brief survey of self-organization in wireless sensor networks. *Wireless Communications and Mobile Computing*, 7(7).

Norman, T. J., Preece, A., Chalmers, S., Jennings, N. R., Luck, M., Dang, V. D., Nguyen, T. D., Deora, V., Shao, J., Gray, A., and Fiddian, N. (2004). Agent-based formation of virtual organisations. *International Journal of Knowledge Based Systems*, 17(2-4):103–111.

Picard, G. and Gleizes, M.-P. (2002). An agent architecture to design self-organizing collectives: Principles and application. In *Adaptive Agents and Multi-Agents Systems*, volume 2636, pages 141–158.

Plaza, E. and McGinty, L. (2005). Distributed case-based reasoning. *Knowl. Eng. Rev.*, 20(3):261–265.

Schillo, M., Bettina Fley and, M. F., Hillebrandt, F., and Hinck, D. (2002). Self-organization in multiagent systems: from agent interaction to agent organization. In *Proceedings of the 3rd International Workshop on Modeling Artificial Societies and Hybrid Organizations (MASHO'02)*, pages 47–56, Aachen, Germany.

Schlegel, T. and Kowalczyk, R. (2007). Towards self-organising agent-based resource allocation in a multi-server environment. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems (AAMAS '07)*, pages 1–8, Honolulu, USA. ACM.

Sierra, C., Rodrguez-Aguilar, J. A., Noriega, P., Esteva, M., and Arcos, J. L. (2004). Engineering multi-agent systems as electronic institutions. *UPGRADE The European Journal for the Informatics Professional*, V(4):33–39.

Sims, M., Corkill, D., and Lesser, V. (2008). Automated organization design for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 16(2):151–185.

Tesauro, G. (2007). Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing*, 11(1):22–30.

Tesauro, G., Chess, D. M., Walsh, W. E., Das, R., Segal, A., Whalley, I., Kephart, J. O., and White, S. R. (2004). A multi-agent systems approach to autonomic computing. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '04)*, pages 464–471, Washington, DC, USA. IEEE Computer Society.

Thompson, J. D. (1967). *Organizations in Action: Social Science Bases in Administrative Theory.* McGraw-Hill, New York, USA.

Tumer, K. and Wolpert, D. (2004). A survey of collectives. In *Collectives and the Design of Complex Systems*, pages 1–42. Springer.

Vazquez, L. E. M. and Lopez y Lopez, F. (2007). An agent-based model for hierarchical organizations. In *Coordination, Organizations, Institutions, and Norms in Agent Systems II (COIN'06 workshops)*, volume 4386 of *LNAI*, pages 194–211. Springer.

Vazquez-Salceda, J., Dignum, V., and Dignum, F. (2005). Organizing multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 11(3):307–360.

Wang, Z. and Liang, X. (2006). A graph based simulation of reorganization in multi-agent systems. In *Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology (IAT '06)*, pages 129–132, Washington, DC, USA. IEEE Computer Society.

Wolpert, D. H. and Tumer, K. (2001). Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2/3):265–279.