Self-Organising Agent Organisations

Ramachandra Kota
School of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ, UK
rck05r@ecs.soton.ac.uk

Nicholas Gibbins
School of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ, UK
nmg@ecs.soton.ac.uk

Nicholas R. Jennings
School of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ, UK
nrj@ecs.soton.ac.uk

ABSTRACT

Self-organising multi-agent systems provide a suitable paradigm for developing autonomic computing systems that manage themselves. Towards this goal, we demonstrate a robust, decentralised approach for structural adaptation in explicitly modelled problem solving agent organisations. Based on self-organisation principles, our method enables the agents to modify their structural relations to achieve a better allocation of tasks in a simulated task-solving environment. The agents reason on when and how to adapt using only their history of interactions as guidance. We empirically show that, in both closed and dynamic organisations, the performance of organisations using our method is close to that of an upper bound centralised allocation method and considerably better than a random adaptation method.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

Algorithms, Experimentation, Performance

Keywords

Autonomic computing, Self-Organisation, Organisations

1. INTRODUCTION

Autonomic systems, capable of self-management, have been advocated as a solution to the problem of maintaining modern, large and complex computing systems [14]. Now, we contend that self-organising multi-agent systems provide a suitable 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 adaptation needs to be performed in a decentralised fashion, so that the ensuing system is robust against failures; again, a characteristic that fits with the multi-agent paradigm [5]. With this motivation, this paper explores the area of self-organisation in agent systems, and particularly focuses on adaptation of the structure in agent organisations.

Cite as: Self-Organising Agent Organisations, Ramachandra Kota, Nicholas Gibbins, Nicholas R. Jennings, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 797–804

In more detail, self-organisation is viewed as *the mechanism or the process enabling the system to change its organisation without explicit external command during its execution time* [7]. Thus, self-organisation can consist of either forming an organisation from disordered entities or rearranging an existing organisation. Such behaviour can be generated in multi-agent systems in several ways [8, 3]. Here, however, we are primarily interested in multi-agent systems that act as cooperative problem solving organisations (i.e those comprising cooperative agents that receive inputs, perform tasks and return results). Hence, we focus on developing self-organisation techniques for such agent organisations.

To date, however, most of the self-organisation mechanisms are not applicable to an explicitly modelled agent organisation because, being based on reactive agents interacting in unstructured ways, they cannot easily be incorporated into agents that are working towards organisational goals. The few mechanisms that do consider agent organisations [12, 10] are centralised in nature, requiring a few specialised agents to manage the adaptation process for all the agents. Though [11] suggest a somewhat distributed method (using a central blackboard), it involves a diagnostic subsystem for detecting faults in the organisation that, in turn, map to some fixed pre-designed reorganisation steps. Such a method is not applicable when all the states of the system cannot be anticipated by the designer. Finally, yet other methods, like organisation self-design [13], achieve self-organisation by dynamically spawning and merging agents in response to the changing requirements. However, since agent-based development of autonomic systems involves modelling the individual components as agents, changing the characteristics of these components may not be possible on all occasions due to physical and accessibility limitations (e.g data-centres located in remote places cannot easily be replicated).

Against this background, we believe decentralised structural adaptation in agent organisations is the most appropriate way of achieving self-organisation. Here, the structure of an organisation is a manifestation of the relations between the agents, which, in turn, determine their interactions. Consequently, adapting the structure involves changing the agent relations, and thereby, redirecting their interactions. For example, consider a query managing agent X repeatedly making queries to a data-centre Y (another agent). Y, in turn, continually passes them to data-centre Z which possesses the types of information needed by X. In such cases, if X and Z recognise this and start interacting directly by forming a relationship between them, it will lead to savings in bandwidth, processor time and memory, while also improving the response time. Also, forming the relation

between X and Z might result in a decrease in the frequency of queries from X to Y. Then, it might be better for X and Y to dissolve their relation in order to improve the overall query processing time of X.

In this vein, [16] identify that structural adaptation is important to improve the performance (in terms of costs and task completion times) of organisations, but do not suggest a way of achieving it. Some work focussing on agent networks [9, 1] deals with agent-based rewiring of acquaintance links towards improving team formation or task allocation, thus somewhat resembling structural adaptation in organisations. However their models assume that only one type of relation can exist in the system and that the number of links possessed by an agent has no effect on its computational resources. These assumptions are unrealistic in cases where agents have to expend resources for managing and delegating tasks based on their relations or links. Moreover, if the adaptation process itself also requires computation, meta-reasoning is needed by the agents to decide 'whether to adapt' (in addition to 'how to adapt') or continue performing the tasks without adaptation. Meta-reasoning, in general, has been explored in a multi-agent systems context [2], but not been applied to self-organisation scenarios. Thus, this aspect is not addressed by any current methods.

In order to address the shortcomings of the existing approaches highlighted above, we present a novel structural adaptation method for agent organisations. Following self-organisation principles, the method is a decentralised and continuous process that is followed by every agent to decide on when and how to adapt its relations, based only on locally available information. Moreover, since it only involves changing the structural relations between the agents, it can even be applied to scenarios where the agents and/or their internal characteristics are not alterable by the system. Thus, our mechanism can serve as a self-management tool similar to those envisioned in autonomic systems. Therefore, our method can be seen to extend the state of the art in terms of structural adaptation mechanisms for agent organisations by being the first that is generically applicable to models with a broad range of inter-agent relations and by addressing the meta-reasoning aspects of adaptation. However, before undertaking this task, we need to have an explicit model of a problem solving agent organisation that can act as the abstract platform on which to base our adaptation mechanism (Sec. 2). By using such a generic platform, instead of focusing on a particular existing system, we can develop a general method that can be applied to a wide variety of applications. We present our adaptation process in Sec. 3, and demonstrate its effectiveness through empirical evaluation in Sec. 4. Finally. we conclude in Sec. 5.

2. THE ORGANISATION MODEL

Organisation modelling involves modelling the agents comprising the organisation, the organisational characteristics, and the task environment. Though there are several existing frameworks for such modelling in the literature [19, 6], we mainly build on the ideas presented by [15] as we found them sufficient for our requirements. In this context, it should be noted that our contribution is the adaptation method and not the organisation model per se. Our adaptation method can be equally applied to other organisation models.

In our model, the agent organisation comprises a group of problem solving, cooperative agents situated in a task environment. By problem solving, we mean agents that receive certain input tasks to achieve, execute these tasks and return the outcomes. Correspondingly, the task environment presents a continuous dynamic stream of tasks that have to be performed. In addition, the environment also has costs associated with passing messages between the agents (communication) and changing their relations (reorganisation).

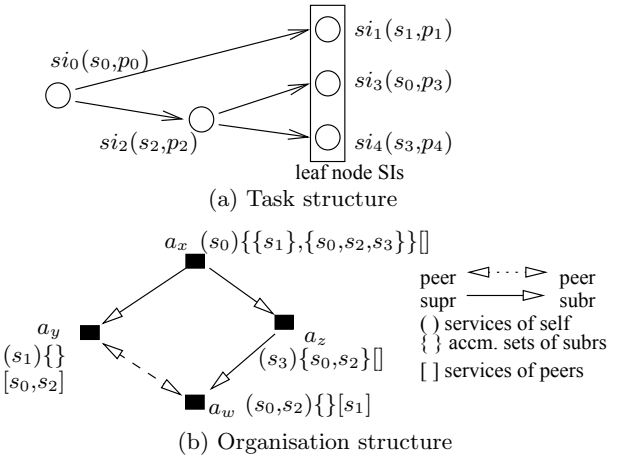


Figure 1: Representation of an example task and organisation

In more detail, the tasks are modelled as workflows composed of a set of service instances (SI), each of which specifies the particular service and the amount of computation required. These SIs need to be executed following a precedence order which is specified in the form of dependency links modelled as a tree. The execution of a task begins at the root node and the task is deemed complete when all its nodes have been executed, terminating at the leaf nodes. Fig. 1(a) shows an example task composed of five SIs ($si_0 \ldots si_4$) each requiring a particular service ($s_0 \ldots s_3$) at a specified amount of computation ($p_0 \ldots p_1$). The required order of execution is shown by the dependency links between the SIs. That is, si_0 needs to be executed first, followed by its child nodes, si_1 and si_2 (which are executed in any order or even in parallel), and so on.

The organisation consists of a set of agents A that provide these services. Every agent is capable of a fixed set of services and possesses a fixed computational capacity. Thus, an agent is of the form $a_x = \langle S_x, L_x \rangle$ where $S_x \subseteq S$ (S is the complete set of services provided by the organisation) and L_x is the agent's capacity defined in terms of available computational units in a time step (these are consumed by the SIs as they are executed). Tasks enter the system at random time-steps and their processing should start immediately. The processing of a task begins with the assignment of the root SI to a randomly selected agent. The agent that executes a particular SI is, then, also responsible for the allocation of the subsequent dependent SIs (as specified by the task structure) to agents capable of those services. Thus, the agents have to perform two kinds of actions: (i) execution and (ii) allocation. Consider an agent executing SI si_0 in Fig. 1(a). After completing the execution, that agent needs to find and allocate appropriate agents to execute si_1 and si_2, the dependent SIs. Moreover, every action has a load associated with it. The load incurred for the execution of a SI is equal to the computational amount specified in its

description, while the load due to allocation (called 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 on a first-come first-served basis, in a single time-step, as long as the cumulative load (for the time-step) on the agent is less than its capacity. If the load reaches the capacity and there are actions still to be performed, these remaining actions will be deferred to the next time-step and so on.

As described earlier, agents need to interact with one another for the allocation of the SIs. These interactions are regulated by the structure of the organisation. This structure is based on the relationships between the agents. We consider three levels of the agent relationship: (i) *acquaintance*, knowing about the presence of, but no interaction, (ii) *peer*, low frequency of interaction, and (iii) *superior-subordinate*, high frequency of interaction. We assume that all agents are acquaintances of each other. In addition, any pair of agents may also have either a peer relation or a superior-subordinate relation between them. These three types of relationships are sufficient to describe the different kinds of interactions possible in a task-allocation setting. In particular, the type of relation present between two agents determines the information that they hold about each other and the interactions allowed between them. We can distinguish between the various relations as follows:

- An agent possesses information about the services that it provides, the services provided directly by its peers and the accumulated service sets of its subordinates. The accumulated service set ($AccmSet$) 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-graphs of agents rooted at each of its subordinates, though it might not know exactly which particular agent is capable of the service.
- During the allocation of a SI, an agent will always try to perform the SI by itself if it is capable of the service and has available computational capacity. Otherwise, it will delegate it to one of its subordinates (which contains the service in its accumulated service set). Only if it finds no suitable subordinate (none of the subordinate sub-graphs are capable of that service), will it try to delegate the SI to its peers. If it is unable to do so either (no peer is capable of the service) it will pass it back to one of its superiors (who will have to find some other subordinate or peer for delegation).

Therefore, an agent mostly delegates SIs to its subordinates and seldom to its peers. Thus, the structure of the organisation influences the allocation of SIs among the agents. Moreover, the number of relations of an agent contributes to the management load that it incurs for its allocation actions, since an agent will have to sift through its relations while allocating a SI. Therefore, an agent with many relations will incur more management load per allocation action than an agent with fewer relations. Also, a subordinate will tend to cause management load more often than a peer because an agent will search its peers only after searching through its subordinates and not finding a capable agent. As all the agents in the organisation are cooperative and work selflessly for the organisation, an agent willingly accepts all SIs delegated by its superiors or peers. Also, the relations are mutual between the agents, that is for any relation existing between two agents, both the concerned agents respect it. In total, the authority relations impose the hierarchical structure in the organisation while the peer relations enable the otherwise disconnected agents to interact closely. Using this model, we abstract away the complex interaction problems relating to issues like service negotiation, trust and coordination. We do so to focus on the essence of self-organisation and to isolate its impact on system performance.

Fig. 1(b) shows an example organisation. The services that an agent can seek from its relations is shown beside it. For example, the accumulated service set $AccmSet_x$ of agent a_x, in turn, contains two sets representing the accumulated service sets of its two subordinates a_y (s_1) and a_z (s_0, s_2, s_3).

Now, given an organisation, the set of agents A can remain static over its existence, or agents may join and leave the organisation. The first kind of organisation, in which the agents are unchanging, is here called *closed*, while the latter is called *dynamic*. In what follows, we apply our method to both. In particular, for dynamic organisations, we focus on those that have some permanent agents to begin with (similar to closed) and some temporary agents who join later, at specified 'start-times', and also leave the organisation at the expiration of their 'life-times'. We adopt this type of dynamic system so that the service set S of an organisation can be kept constant (the temporary agents will offer services chosen from the same S as the permanent ones). In this way, our method can focus solely on the changes to the overall capacity (resulting from the temporary agents) instead of the service discovery aspect that might have been needed[1]. Consequently, these dynamic organisations represent distributed systems in which additional resources might be added to tackle the workload and withdrawn later on.

2.1 Organisation Performance Evaluation

The performance of a computing system denotes how well it performs its tasks. In terms of an agent organisation, the performance measure can be abstracted to the profit obtained by it. In our model, the profit is simply the sum of the rewards gained from the completion of tasks when the incurred costs have been subtracted. In more detail, the cost of the organisation is based on the amount of resources consumed by the agents. In our case, this translates to the cost of sending messages (communication) and the cost of changing relations (reorganisation) between the agents. Thus, the cost of the organisation is:

$$cost_{ORG} = C. \sum_{a_x \in A} c_x + D.d \qquad (1)$$

where C is the communication cost coefficient representing the cost of sending one message and D is the reorganisation cost coefficient representing the cost of changing a single relation. c_x is the number of messages sent by agent a_x and d is the number of relations changed in the organisation.

As stated earlier, agents have limited capacities and their computational load cannot increase beyond this capacity. The load l_x on agent a_x in a given time-step is:

$$l_x = \sum_{si_i \in W_{x_E}} p_i + M \sum_{si_j \in W_{x_F}} m_{j,x} + R.r_x \qquad (2)$$

where p_i is the amount of computation expended by a_x for executing SI si_i, $m_{j,x}$ is the number of relations considered by a_x while allocating SI si_j, W_{x_E} is the set of SIs (possibly

[1]We intend to look at other types of dynamics in the future.

belonging to several tasks) being executed by a_x, W_{x_F} is the set of SIs being allocated by a_x, and M is the 'management load' coefficient denoting the computation expended by an agent to consider one of its relations while allocating a single SI. In this way, it represents the computational complexity resulting from the relations of an agent. Similarly, R is 'reorganisation load' coefficient, denoting the amount of computational units consumed by an agent while reasoning about reorganisation with another agent and r_x is the number of agents that a_x initiated reorganisation deliberation in that time-step. Thus, this represents the excess load caused by meta-reasoning about reorganisation. Since, the load l_x of a_x cannot exceed its capacity L_x, any excess SIs will be waiting for their turn, thus delaying the completion time of the tasks. The rewards obtained by the organisation depend on the speed of completion of tasks. In particular, a task w completed on time accrues the maximum reward b_w which is the sum of the computation amounts of all its SIs: $b_w = \sum_{i=0}^{|si_w|} p_i$, where si_w is its set of SIs. For delayed tasks, this reward degrades linearly with the extra time taken until it reaches 0: $reward_w = b_w - (t_{taken} - t_{reqd})$, where t_{taken} represents the actual time taken for completing the task, while t_{reqd} is the minimum time needed. Thus, the total reward obtained by the organisation is the sum of the rewards of the individual tasks completed by the organisation:

$$reward_{ORG} = \sum_{w \in W} reward_w \quad (3)$$

where W is the set of all tasks. The organisation's profit is:

$$profit_{ORG} = reward_{ORG} - cost_{ORG} \quad (4)$$

Thus, for higher profits, the reward should be maximised, while the cost needs to be minimised. It is important to note that the agents only have a local view of the organisation. They are not aware of all the tasks coming in to the organisation (only those SIs allocated to them and their immediate dependent SIs) and neither are they aware of the load on the other agents. In spite of this incomplete information, they need to cooperate with each other to maximise the organisation profit by maintaining the most useful relations which lead to faster allocation and execution of tasks.

3. STRUCTURAL ADAPTATION

This section details our work on developing a self-organisation based structural adaptation method that can be employed continually by all the agents in a problem-solving organisation. It uses the history of agent interactions only, since we do not assume that agents possess any information about the tasks coming in the future. First, we present the basics of the method and then show how to deal with the temporary agents of dynamic organisations.

3.1 The Algorithm

We present the mechanism, in a pseudocode form in Algorithm 1, for how an agent a_x should reorganise at a given time-step. The first component (line 1) refers to the meta-reasoning aspect of choosing the particular acquaintances for initiating the reorganisation process. The second component (lines 3–9) explains how it should adapt its relation with one such acquaintance a_y.

We formulate this part using a decision theoretic approach since it provides us with a simple and suitable methodology for representing adaptation in terms of actions and utilities. We denote the actions available to a pair of agents as those

```
1 Chosen ← selected from the acquaintances set of a_x;
2 foreach a_y ∈ Chosen do
3 |   Actions ← possible_actions(x, y);
4 |   U_{x,y} ← ∅;
5 |   foreach e ∈ Actions do
6 |   |   U_e ← calculate_utility_{x,y}(e);
7 |   |   U_{x,y} ← U_{x,y} ⋃ U_e;
  |   end
8 |   e_best ← argMax(U_{x,y});
9 |   take action e_best with a_y;
  end
```

Algorithm 1: Reorg. method in terms of agent a_x

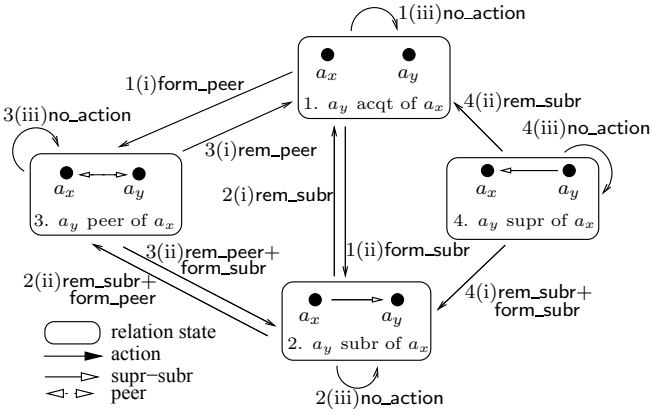


Figure 2: State transition diagram

changing the relation between them. Consequently, the set of actions available to a pair depends on their relation (line 3). In our model, for every pair of agents, the relation between them has to be in one of the states— purely acquaintance relation, peer relation or authority relation. For each of these states, there are 3 possible choices of action available to the agents as shown in Fig. 2. For example, action 1(ii) (form_subr$_{x,y}$) denotes that a_x and a_y take the action of making a_y a subordinate of a_x and transition from state 1 to 2. A transition from state 2 to 4 is not needed because it is equivalent to the transition from 4 to 2, by interchanging a_x and a_y. Similarly, transitions from 1 to 4 or between 3 and 4[2] are not required. If there were more types of relations in the model, there would be correspondingly more states and transitions to represent them.

As can be seen, these actions are composed of four atomic types— form_peer, rem_peer, form_subr and rem_subr, which translate to forging and dissolving the peer or authority relations (as agents are acquaintances of each other, by default). Obviously, each of these actions has to be jointly performed by the two agents involved in changing the relation. The utility of performing an action (U_e in line 6) is given by value function V associated with the relation. Since our environment is characterised by various factors, like the costs and the load on the agents, V will have multiple attributes to represent them. In terms of two agents, a_x and a_y, the

[2]In state 4, 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 are not permitted). Hence, it dissolves its relation with its immediate superior in the authority chain a_y and goes to state 1 w.r.t a_y and then forms a relation with a_z.

five attributes that will constitute V are: (i) change to the load on a_x, (ii) change to the load on a_y, (iii) change to the load on other agents of the organisation, (iv) change to the communication cost and (v) reorganisation cost. Moreover, 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, V can be represented simply as a sum of the functions pertaining to the individual attributes:

$$V = \Delta load_x + \Delta load_y + \Delta load_{OA} + \Delta cost_C + \Delta cost_R \quad (5)$$

In this way, depending on the state, the agents jointly calculate the expected utilities for the possible actions using the value function (which are stored in $U_{x,y}$ at line 7), and then choose the action giving the maximum expected utility (line 8). Being cooperative, the agents do not have conflicts as the value corresponds to the social utility of the relation to the organisation and not to the individual agents. 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 Eqn. 5. In the case of the composite actions (like rem_subr+form_peer) the value 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 is jointly conducted by the agents with each of them supplying those attribute values accessible to them.

To understand further, let us look at the utility calculation (Eqn. 5) for the action $\mathsf{form\_subr}_{x,y}$ when a_x and a_y are just acquaintances (state 1). Here, $\Delta load_x$, representing the estimated change to the load on a_x, is calculated by considering that a new subordinate a_y will lead to an increase in the management load on a_x every time it tries to allocate a SI. This is quantified as:

$$\Delta load_x = -M * Asg_{x,total} * filled_x(t_x^{total})/t_x^{total} \quad (6)$$

where $Asg_{x,total}$ denotes the total number of SIs allocated by a_x, t_x^{total} denotes the total number of time-steps that a_x has been in existence, while $filled_x(t_x^{total})$ represents the number among those that a_x's capacity was filled with load and some SIs were pending. By multiplying this value with M, it represents the additional load that would have been put on a_x had a_y been a subordinate since the beginning. The negative sign indicates that this value represents addition to load, and thereby, a decrease in utility. In a similar fashion, the second term $\Delta load_y$ is calculated by estimating the possible increase in the load on a_y had a_x been its superior since the beginning. For the third term, $\Delta load_{OA}$, the estimation is carried out by assuming that, had a_y been a subordinate of a_x, all those allocations that started at a_x and ended at a_y via a delegation chain involving other agents, would have been allocated directly. Therefore, the load on the intermediary agents would have been less (this value can be calculated by a_x as it gets back information about the delegation chain of each of the SI allocated by it). Similarly, the fourth term $\Delta cost_C$ is also estimated, while the last term, the reorganisation cost ($-D$) is a known constant (the minus sign, again signifies a decrease in utility).

In this way, using our mechanism, every pair of agents can jointly evaluate the utility for taking any of the possible actions towards changing their relation, at any time-step. Thus, this continuous adaptation of the relations helps in the better allocation of SIs amongst the agents as they will maintain relations with only those agents with whom they require frequent interactions.

In the ideal scenario, at line 1, all acquaintances of an agent will be chosen for reasoning about adaptation. However, as the computation required for these utility calculations and reasoning depends on R (Sec. 2.1), it need not be negligible and might exhaust the computational capacities of the agent. Thus, when R cannot be ignored, an agent will have to smartly select the acquaintances for *Chosen* in line 1. Thus, effective meta-reasoning emerges as an important aspect of the adaptation process.

In our case, this problem boils down to the following— at any given time-step, an agent should decide on how many and which agents to select for initiating reorganisation procedures. This can be viewed as a form of the well-known *coupon collector's problem* [18] and, therefore we explore a simple randomised approach that is typically used for such problems. In the coupon collector's problem, there are n types of coupons and an infinite number of coupons for each type. At each trial, a coupon is chosen at random. We can map this problem to our scenario by considering every agent to be the collector, and all its acquaintances (including the peers, superiors and subordinates) as the coupons. Also, in our case, there can be several trials in a single time-step.

Now, if X is the number of trials such that at least one coupon of each type is collected, then the expectation of X is: $E(X) = nln(n) + O(n)$ [18]. This assures us that even when chosen randomly, on average, all acquaintances of an agent will be picked up for reorganisation deliberation in a given period of time (for 20 acquaintances, this translates to approximately just 80 trials). Therefore, an agent can just randomly choose k acquaintances at a time-step (in line 1). Moreover, this k can be varied according to the situation. When an agent has free capacity that will otherwise be wasted, k should be such that the whole of the remaining capacity is utilised for reorganisation. However, even when the agent is overloaded, reorganisation might be necessary. On such occasions, k is based on the percentage of successful reorganisations in the previous time-step. In more detail, at a time t, k is determined as:

$$k_t = max\{1, (L_x - l_x)/R, acqts_x * changed_{x,t-1}/k_{t-1}\} \quad (7)$$

where $acqts_x$ represents the number of acquaintances of a_x, $changed_{x,t-1}$ denotes the number of relations of a_x changed in the previous time-step, and k_{t-1} is the k value used in the previous time-step. The minimum possible value of k is limited to 1, so that at least one acquaintance is considered for reorganisation in any time-step. In this way, free capacity is never wasted and, at the same time, an agent will carry out reorganisation even when it has a huge number of pending SIs by adapting k according to its need for reorganisation.

3.2 Dealing with Dynamic Organisations

Having outlined the fundamental part of the structural adaptation method, we now discuss how it deals with dynamic set of agents in an organisation. When a new agent joins the organisation, it needs to be assimilated into the structure by the existing ones. However, for an agent to form a relation with a new agent, it has to be able to predict how useful the new agent will be and in what type of relation. This is not straightforward as there are no past interactions with the new agent on which to base any utility calculations (as required by our method). Therefore, the agent is faced with an explore versus exploit trade-off: whether to

explore by forming an authority or a peer relation with a new agent, or to reorganise with the past agents only by *exploiting* the known information about them. This choice could be tackled by employing specially designed 'explorer agents', whose sole task is to monitor the performance of all the agents (including the new ones)[17]. However, we do not use such an approach because it requires special agents and that contradicts our self-organisation principles. Thus, our intention is to imbibe the adaptation method into the task-solving agents without needing any external help.

Against this background, we find that the principle behind 'Win Or Learn Fast' [4] is well suited to our problem. The WoLF principle is— '*learn quickly while losing, slowly while winning*'. In our context, an agent can be considered winning if it has unused capacity and losing otherwise (when it has a pending queue of SIs). Therefore, an agent that is not overloaded will only follow Algorithm 1 by ignoring the new agents joining the system. However, an agent with pending SIs will actively seek new subordinates to be able to improve its delegation of SIs. This addition to our method is presented as a pseudocode in Algorithm 2.

```
1 if W_{x_P} ≠ ∅ then // Set of pending SIs of a_x
2 |  s ← arg-MaxOccuring{W_{x_P}} AND ∉ AccmSet_x;
3 |  A_s ← agents providing service s;
4 |  a_y ← randomly chosen from A_s;
5 |  form_subr_{x,y};
  end
```

Algorithm 2: In terms of agent a_x applying WoLF

In more detail, an agent, when overloaded (checking in line 1), identifies which particular service occurs the most in its waiting list that is not supplied by any of its current subordinates (line 2). Then, in line 3, it searches through all of its acquaintances (including the newly joined agents) for those offering that particular service. Finally, in lines 4–5, it forms a superior-subordinate relation with one such randomly chosen agent. As a result, new agents will be assimilated quickly by the existing agents into the structure. Moreover, these new agents will end up forming the relations where they are most needed, thereby leading to a more equitable distribution of load across the organisation. In addition, a new agent offering services which are not much in demand, will be ignored (as the agents offering those services are winning anyway) and thus not add any unnecessary management load. In contrast, when an agent leaves, the others can easily reorganise using our method in Algorithm 1 without needing any such additional step.

4. EMPIRICAL EVALUATION

We demonstrate the effectiveness of our method through empirical evaluation. We first describe the setting that we used for experimentation and then present the obtained results.

4.1 Experimental Setup

To determine the effectiveness of our method (which we call `k-Adapt`), we compare its performance with two other intuitive methods— `Central` and `Random`, that act as the benchmarks. We also compare with a few variations of `k-Adapt` to show the importance of all the components of our algorithm. All of these methods are described below:

Central: This is a centralised allocation mechanism containing a central repository that maintains information about the service sets and loads of all the agents in the organisation, and is accessible without cost to any agent. The agents do not need to maintain any relations; whenever an agent needs to allocate a SI, it looks up the repository seeking the most suitable agent (capable of the service and having maximum free capacity at the time) and allocates to it. Thus, all allocations are one-step direct delegations, and the agents do not use up any capacity for allocation. This method gives an upper bound on the performance of an organisation, but is not a practical or robust solution to the problem because it involves maintaining an up-to-date central repository with costless and instantaneous access to all agents.

Random: In terms of Algorithm 1, this strategy involves an agent randomly choosing some of its acquaintances for adaptation (line 1), and then randomly choosing a reorganisation action (line 8). For a fair comparison, the rate of change is adjusted so that the number of relations in the structure is roughly equal to that produced by our method. Thus, this method represents a random structural adaptation strategy which does not involve any reasoning and constitutes the lower bound.

free-Adapt: For this method, the reorganisation load coefficient R is set to 0. Thus, it is the same as `k-Adapt` but differs only in line 1, where all the acquaintances are chosen for reorganisation instead of just k. This makes it a theoretical upper bound for the performance of our method.

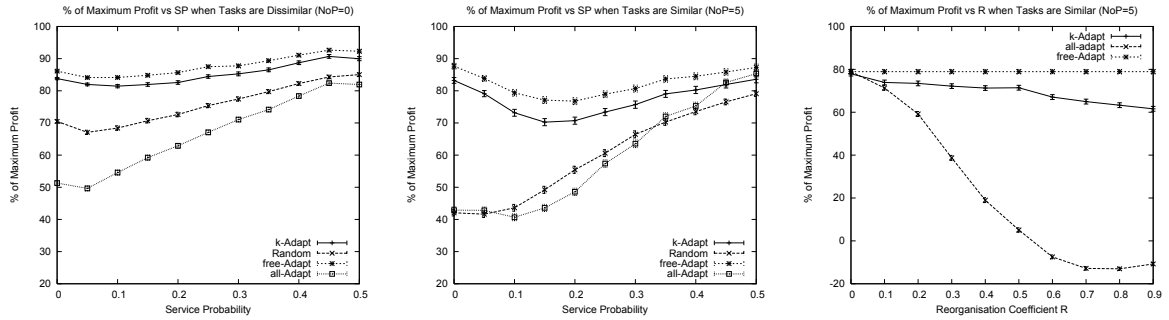
all-Adapt: Same as `free-Adapt`, differing only in R, which is set to the same value as in `k-Adapt` and not 0.

nowolf-Adapt: This method is used for comparison in dynamic organisations. It is similar to `k-Adapt` but without including the WoLF component presented in Algorithm 2.

While the profit obtained by the organisation ($profit_{ORG}$ in Eqn. 4) represents the organisation's performance, there are two independent simulation variables that are of interest:

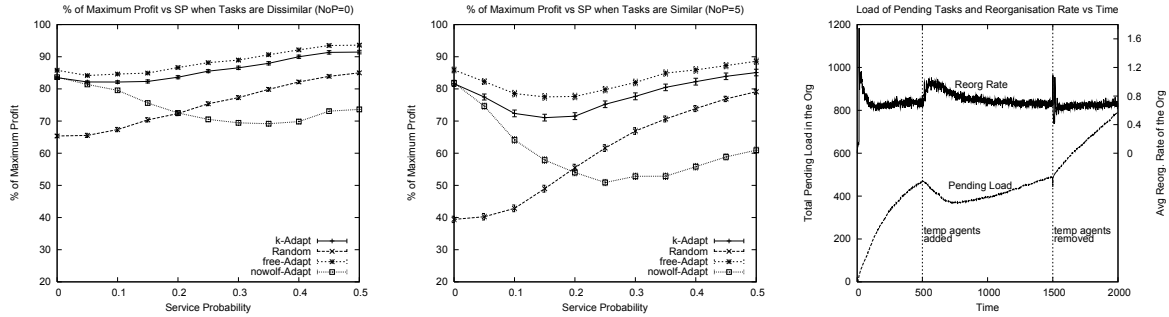
Distribution of services across agents: This is a relevant parameter because the significance of the organisation structure is greater when the agents are heterogeneous. In such a case, an efficient structure will need to connect every agent with all those agents providing services relevant to it and alongside help in load distribution. In contrast, for homogeneous agents, load distribution is the only feature that can be influenced by the structure. We distributed the services among the agents using a parameter 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. When it is 1, every agent is capable of every service. Since, the services are allocated on the basis of a probability, there is always randomness in the way they are allocated to the agents. In our experiments, we vary SP from 0 to 0.5 only (since we verified that beyond 0.5, when the agents are quite homogeneous, the structures did not influence the performance significantly).

Similarity between tasks: This is also an interesting parameter because when tasks are similar, the structure should be able to adapt to the recurring task structures, thereby increasing the efficiency of the organisation. We determine the similarity between the tasks belonging to a simulation run on the basis of what we call *patterns*; stereotypical task components used to represent frequently occurring combinations of SIs and dependency links. Like tasks, patterns are also composed of SIs, but are generally smaller in size. Instead of creating tasks by randomly generating SIs and creating dependency links between them, tasks can be constituted by connecting some patterns by creating dependency links between the SIs belonging to the patterns. In this way, the dependencies between the SIs may follow some



(a) Dissimilar Tasks (NoP=0) (b) Similar Tasks (NoP=5) (c) Profit as R increases

Figure 3: Average Organisation Profit for Closed Organisations



(a) Dissimilar Tasks (NoP=0) (b) Similar Tasks (NoP=5) (c) Pending Load & Reorg. Rate

Figure 4: Average Organisation Profit for Dynamic Organisations

frequent orderings (resulting from the dependencies internal to a pattern occurring in several tasks) and some random dependencies (due to the dependencies created between 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 consider two scenarios: (i) completely dissimilar tasks ($NoP = 0$) and (ii) highly similar tasks ($NoP = 5$).

All our experiments comprise 1000 simulation runs for every data point to achieve statistically significant results. All the results are shown with 95% confidence intervals (the errors bars are very close to the marking symbol), obtained by multiplying the standard error by 1.96 (z-test). For every simulation, the set of agents and services is first generated and then the services are assigned to the agents on the basis of SP. Next, the set of tasks is generated using NoP. In our experiments, we use a maximum of 20 initial agents in the organisation which, in turn, faces 1500 tasks over 2000 time-steps to constitute one simulation run. Furthermore, we set C at 0.25 and M at 0.5 (so that any allocation process will take up at least half a computational unit). Also, we set R at 0.25 and D at 1. The maximum size of a pattern is limited to 8 so that, on average, three patterns are required to compose a task (which can have a maximum of 25 SIs). We observed broadly similar patterns with other parameter settings. Finally, the set of agents A, is kept constant for closed organisations, while in the dynamic case, a randomly chosen number of temporary agents are added, as described in Sec. 2. The results shown here are of experiments where the start-times and life-times are chosen from a uniform distribution. However, we also conducted experiments with a combination of distributions for start-times (uniform and normal) and life-times (normal and geometric) and found the resultant trends to be same as these. We present the results as graphs plotting the percentage of the maximum profit obtained for the strategies over an increasing SP along the x-axis (increasing similarity of the agents). The maximum profit is given by the profit obtained by `Central`.

4.2 Results: Closed Organisations

In both the scenarios with dissimilar (Fig. 3(a)) and similar tasks (Fig. 3(b)), `k-Adapt` performs consistently better than `Random`. The difference in their performance narrows down (from the highest of 40% of profit to 10%) as the similarity of agents increases because a smart method is correspondingly less useful when all the agents are homogeneous, as the significance of the structure itself diminishes. Also, we see that `k-Adapt` and `free-Adapt` perform better when $SP = 0$ than for slightly higher values of SP because, as SP increases and more agents are capable of a given service, `Central` continues performing perfect allocations (as it has up-to-date information about loads on all agents), while the agents in the organisations using our method have no way of knowing which relations have free capacities. However, the performance increases for higher values of SP because the average capacity available for any given service becomes larger as agents are capable of more services, thus leading to better task completion times. This is also the reason why `Random` improves with increasing SP. We also conducted experiments by varying R from 0 to 0.9 (see Fig. 3(c)) and found that the fall in the performance of `k-Adapt` is gradual and minimal, while it is drastic in `all-Adapt`. In fact, for higher values of R, the profit of `all-Adapt` goes below 0, meaning the cost is more than the reward obtained. This shows that meta-reasoning is a crucial aspect in an adaptation process and cannot be ignored.

4.3 Results: Dynamic Organisations

In the dynamic scenarios, we again find that `k-Adapt` performs considerably better than `Random` when tasks are both dissimilar (Fig. 4(a)) and similar (Fig. 4(b)). More importantly, unlike `k-Adapt`, `nowolf-Adapt` degrades rapidly as the similarity between agents increases. This shows that the WoLF principle is very useful for assimilating new agents into the organisation and maintaining the performance.

Furthermore, Fig. 4(c) gives us an insight into what is happening to the organisation when the agents are added and removed. For this experiment, we fixed the start-time at 500 and life-time at 1000 for the temporary agents. The graph shows the sum of the computations of all pending SIs in the organisation (left y-axis) across the time duration of the simulation, and shows the corresponding reorganisation rate in terms of the number of relations changed in a time-step (right y-axis). We observe a gradual fall in the load starting at time=500 corresponding to when temporary agents are added. Also at time=1500, there is a quick drop and immediate increase because, when the temporary agents leave, the SIs pending at them are reassigned to the permanent agents. This reassignment requires at least a time-step after which only they are visible as pending load again. Also, the rate of growth of pending load is higher once the agents leave (as seen by the higher gradient). Looking at the reorganisation rate, we find that it is high in the beginning and then settles down to an almost uniform rate. Later, there is a sudden jump in the rate when the agents are added and this gradually falls back to the earlier value at around time=700. This shows that our adaptation process is able to reach its earlier stable state in reasonable time. As expected, we also find another blip in the rate when the agents are removed. This time, it settles much more quickly as the permanent agents are able to easily reform the older structure that existed prior to the addition of the temporary agents.

In summary, we find that the performance of our adaptation method is around 80% of the centralised allocation method which is, on average, 15% better than a random reorganisation approach (reaching up to a maximum of 40%). Also, we have seen that both the high-level aspects of our method— the randomised approach for selection of agents for reorganisation and the WoLF principle for temporary agents— are crucial for this good performance.

5. CONCLUSIONS AND FUTURE WORK

This paper addresses the problem of developing decentralised structural adaptation methods for problem solving agent organisations based on the paradigm of self organisation. Using a simple organisation model as a framework, we presented a structural adaptation method that can be applied individually by all the agents in order to improve the organisation performance. Using the method, a pair of agents estimate the utility of changing their relation and take the appropriate action accordingly. Moreover, our method also enables an agent to meta-reason about when and with whom to initiate reorganisation. A useful feature of our adaptation method is that it works purely by redirecting agent interactions (via the organisation structure) and does not entail any modifications to the agents themselves or their internal characteristics. Since the adaptation method is purely agent-based, decentralised and continuous over time, it satisfies the principles of self-organisation discussed in Sec. 1. Thus, it can be used by the individual components of a distributed system to self-manage by helping them adapt their interactions with the other components in a local and robust way. We have empirically shown that our method performs at 80% of a centralised omniscient method and is significantly better than a random method.

In future work, we intend to test the applicability of our method by using real-data from existing distributed computing scenarios. We also plan to improve our method for scenarios where agents might be gaining new services and losing old services, thus reflecting systems whose capabilities are upgraded with time.

6. REFERENCES

[1] S. Abdallah and V. Lesser. Multiagent reinforcement learning and self-organization in a network of agents. In *Proc of the 6th AAMAS*, pages 172–179, USA, 2007.

[2] G. Alexander, A. Raja, E. H. Durfee, and D. J. Musliner. Design paradigms for meta-control in multi-agent systems. In *Proc. of AAMAS Workshop on Metareasoning in Agent-based Systems*, pages 92–103, USA, 2007.

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

[4] M. Bowling and M. Veloso. Rational and convergent learning in stochastic games. In *Proc. of the 17th IJCAI*, pages 1021–1026, USA, 2001.

[5] T. De Wolf and T. Holvoet. Towards autonomic computing: agent-based modelling, dynamical systems analysis, and decentralised control. In *Proc. of the 1st Intl Workshop on Autonomic Comp. Princ. and Arch.*, Canada, 2003.

[6] S. A. Deloach, W. H. Oyenan, and E. T. Matson. A capabilities-based model for adaptive organizations. *AAMAS*, 16(1):13–56, 2008.

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

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

[9] M. E. Gaston and M. desJardins. Agent-organized networks for dynamic team formation. In *Proc. of the 4th AAMAS*, pages 230–237, The Netherlands, 2005. ACM.

[10] M. Hoogendoorn. Adaptation of organizational models for multi-agent systems based on max flow networks. In *Proc. of the 20th IJCAI*, pages 1321–1326, India, 2007.

[11] B. Horling, B. Benyo, and V. Lesser. Using self-diagnosis to adapt organizational structures. In *Proc. of 5th Intl. Conf. on Autonomous agents*, pages 529–536, NY, USA, 2001.

[12] J. F. Hubner, J. S. Sichman, and O. Boissier. Using the MOISE+ for a cooperative framework of MAS reorganisation. In *Proc. of the 17th Brazilian Symposium on AI*, volume 3171, pages 506–515. Springer, 2004.

[13] S. Kamboj and K. S. Decker. Organizational self-design in semi-dynamic environments. In *Proc. of the 6th AAMAS*, pages 1220–1227, Honolulu, USA, 2007.

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

[15] R. Kota, N. Gibbins, and N. R. Jennings. Decentralised structural adaptation in agent organisations. In *Proc. AAMAS Workshop on Organised Adaptation in Multi-Agent Systems*, pages 1–16, 2008.

[16] P. Mathieu, J.-C. Routier, and Y. Secq. Principles for dynamic multi-agent organizations. In *Proc. 5th Pacific Rim Intl Workshop on MAS*, pages 109–122, UK, 2002.

[17] E. M. Maximilien and M. P. Singh. Multiagent system for dynamic web services selection. In *Proc. of 1st Workshop on Service-Oriented Computing and Agent-Based Engineering*, pages 25–29, Netherlands, 2005.

[18] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, New York, USA, 1995.

[19] J. Vazquez-Salceda, V. Dignum, and F. Dignum. Organizing multiagent systems. *AAMAS*, 11(3):307–360, 2005.