Dynamic Organization of Multi-Agent Systems

Philippe Mathieu, Jean-Christophe Routier, and Yann Secq

Laboratoire d'Informatique Fondamentale de Lille
Cité Scientifique 59655 Villeneuve d'Ascq Cedex
{mathieu,routier,secq}@lifl.fr
http://www.lifl.fr/SMAC

Abstract. Many models of organizations for multi-agent systems have been proposed so far. However the complexity implied by the design of social organizations in a given multi-agent system is often not mentioned. Too little has been said about rules that must be applied to build the architecture of acquaintances between agents. Moreover, tools for managing the dynamic evolution of organizations are seldom provided in current framework propositions.
In this paper we discuss self-adaptation of organizations in multi-agent systems according to the dynamic of interactions between agents. Starting from a default organization, the architecture of acquaintances evolves autonomously depending on messages flow in order to improve the global behaviour of the system. We propose three principles that can be applied to adapt the organization: "have a good address book", "share knowledge", "recruit new able collaborators".
These principles have been applied in our multi-agent platform called MAGIQUE.

1 Introduction

Multi-agent systems can be seen as societies of interacting agents. This notion of interaction, which allows agent to find each other and then to exchange information, is a central point for the design of multi-agent applications. Some methodologies have been proposed, and they always identify the need that agents have to *get in touch* with other agents, but they seldom provide guidelines to design the acquaintances structure. The GAIA[13] methodology, for instance, identify this stage as the *acquaintance model*, which is defined as follow :

> *An agent acquaintance model is simply a graph, with nodes in the graph corresponding to agent types and arcs in the graph corresponding to communication pathways. Agent acquaintance models are directed graphs, and so an arc $a \rightarrow b$ messages to b, but not necessarily that b will send messages to a. An acquaintance model may be derived in a straightforward way from the roles, protocols, and agent models.*

We see that this definition just defines what we could call the *natural* notion of acquaintance. The notion of organization is even not clearly identified. In another work [5], it is stated that :

> *an Interaction Model describes the responsibilities of an agent class, the services it provides, associated interactions, and control relationship between agent classes.*

Again, this is just a way to express that agents interact and so need to have some communication paths to exchange information. Others methodologies [2, 10], often state the same kind of concepts but seldom identify that the acquaintance structure is a first-class citizen of MAS entities.

Some works highlight the importance of the notion of organization in multi-agent systems[14, 6]. Unfortunately, those works seldom reify this notion. For example, the Contact-Net Protocol [12], which is based on a market-type system where providers make propositions to requesters, is one kind of organization : it provides a mean to find the "best" acquaintance for a given task. The Aalaadin [7] model, which relies on the idea that agents are identified by roles they hold within some groups, is another kind of organization. Others works emphasized the notion of hierarchy : the Magique [11] model propose a hierarchical structure where agents have a boss and can manage a team. Finally, the holonic approach [3], defines the notion of holarchy to organize holons. Roughly speaking, holons must conform to some contracts that allow dynamic agent (holon) creation through the aggregation of a set of sub-agents (sub-holon). Those works, even if they do not emphasize the same nature of social structure, promote the same idea : organizations are backbones of multi-agent systems.

Building an organization to optimize agent interactions is not straightforward : how should we spread functionalities among agents, and how is it possible to reduce the cost of communication, and overall how can the system deals with agents that freely leave or join it? Lastly, how organizations can deal with the ever-changing flow of agents interactions? This paper postulates that this complexity should not be exclusively addressed by the multi-agent system designer. Organizations infrastructures should provide default behaviours to dynamically optimize communications flow, in order to lower the number of messages that are exchanged, or to improve the quality of service. Too little works [9, 4] have been done in this direction.

In the first section, we describe the needs to have an adaptive organization. We first present static organizations and their limitations, then we study how social organizations deal with those problems before we apply their solutions to multi-agent systems. The second section introduces the MAGIQUE multi-agent framework and uses it to illustrate dynamic organizations through some simple experiments described in section three.

2 Adapting the Architecture of the Organization

Before we consider how to adapt the organization of a multi-agent system, some problems with predetermined static structures must be considered. We will then propose some general strategies to tackle these problems.

2.1 Some problems with static organizations.

One of the first problem, and probably the basic one, is to determine how acquaintances are created? That is, how an agent can have information about the existence of another able agent. One solution of course, is that this can be predetermined and established by the multi-agent system designer, but this is not a satisfactory answer. Firstly, how

should this designer proceed to choose the most fitted acquaintance architecture, which methodology must be applied, if there exists any really convenient. And secondly, what about systems where new agents appear or what happens when the “able acquaintance” is removed from the system, or becomes unavailable, because of a network failure for example?

A second problem is more connected with the distribution of the skills over the agents and is related with performance issues similar to load balancing. How can the system be organized in such a way such that no agent becomes a critical overloaded resource[8]? This implies that even if an organizational structure has been chosen, this is not enough. You need to choose how the skills are distributed among the agents. It is of course difficult if not impossible, to give universal rules to do this. It would be better if one agent does not become a bottleneck in the system simply because he is the only one able to provide a too often required service. In this situation you probably prefer the service to be provided by several agents. Of course, this is not always appropriate, in the case of some certification service for example. But when it is, how could it be predetermined ? It is not necessarily obvious which service will be critical (in term of overloading) and, even if you give such a service to several agents, how to ensure that one of the able agents will not be overused and others ignored.

Lastly, we will consider a situation which is not so far from the previous one but where we consider the “client of service” point of view rather than that of the service provider. One agent may often have to use some given service for which he must make requests to an able agent. In this case, even if the service provider agent is not overburdened, the client agent will probably be penalized by too many requests, at least because of the communications. It would have been better, when designing the system, to qualify this agent with the service, or to allow the agent to dynamically acquire it.

Aware of these problems, a multi-agent system designer will take them into account and try to anticipate them and he will attend to limit them. He could succeed in that, but what happens in the context of dynamic multi-agent systems, where agents can freely join or leave the system ? This implies that some services will become available at some time and unavailable at other. Agents must adapt themselves to this dynamic environment. The designer cannot predetermine those situations. Therefore the only thing he can do is to prepare his agents in such a way that they can adapt autonomously to the changes that occur within their environment. In consequence, general strategies must be given, we will discuss some of them in the following.

2.2 How do social organizations manage these problems ?

The problems we have raised in the previous section are not peculiar to multi-agent systems but are general to social organizations, where members can be persons or companies.

In every social structure, the problem of finding the “right person for the job” appears. Often this “right person” is not known *a priori* and it is necessary to use known acquaintances to find who it is. But, of course, this may be a source of problems. You do not necessarily want to use some go-between that can know what you want from the “right person” and make use of this information. Moreover, this can have a cost

since the middleman can ask for a payment only because he has helped you to get in touch with the good person. Therefore after some time, when information can have been gathered you try to reach the right person directly.

The problem of overloaded resources exists too. The more able a person or a company is, the more probable it is that she or he will be overburdened (in fact this is often considered as a symptom of competence). And then the delay before you benefit from its service increase. In this case, the too much appealed resource must often find a way to speed up its answer. Else, in the case of a company for example, clients will be seeking an equivalent service somewhere else.

If you consider the client side, making too often requests to some critical resource is a major drawback which has a cost. Either a time cost because client must wait for the availability of the resource, or a money cost because clients pay for the service. Therefore, when it is possible, clients try to go round this dependence.

In these three cases, the problem of cost or efficiency appears. In social organizations, there is a trend to aim at better efficiency. This trend can be natural – we all have tendency to apply the law of least effort –, or economical by trying to reduce cost – unless the intent is to increase profit? –.

We have identified three principles that can be used to improve the global behaviour and that implies a dynamical organization of the social structure :

1. having a good address book,
2. sharing knowledge (or selling it...),
3. recruiting new able collaborators.

The first principle deals with the first of the three previous problems. It may seem that this principle could have been called “remove the go-betweens”, however this must be moderated. Indeed, creating new (social) links has a cost and it is not appropriate to always go round the go-between. This one may know his job and his offer for a given service can change because he has found of a better provider. In such a case the use of the go-between would have been beneficial. In consequence, “having a good address book” does not mean always removing the go-between, but rather knowing when to use him and when not.

The second and third principles are rather means to tackle second and third problems and more generally to improve efficiency by reducing the time necessary for a service request to be treated. When a service company is overused, in order not to lose client, it will probably recruit able collaborators. In the same way, when the company needs a new skill, it can recruit new collaborators with the required competence. Or, consider a craftsman with too many orders, he will take one or more apprentices and train them. This is a combination of the two principles, even if it is more of the “sharing knowledge” since the intention is that, after his or her training, the apprentice becomes a new resource. Of course, once again, recruiting or teaching/learning knowledge has a cost and can not be applied every time.

2.3 The three principles applied to multi-agent systems

These three principles can be applied to achieve a self organization of the social structure in multi-agent systems. By applying them, we want an evolution of the acquain-

tance structure and the distribution of skills in order to reduce, firstly, the number of messages exchanged in the system and, secondly, the time necessary for a service request to be treated.

According to these principles, we start from a predetermined organization, where the agents have default acquaintances and where skills (or services) are more or less arbitrarily distributed over the agents. The idea is to have an evolution of the structure of acquaintances where the natural links are favoured at the expense of predefined ones.

Of course the major benefit is for the designer of the multi-agent system who can prepare his multi-agent system as it seems the most fitted and then rely on these principles to adapt the efficiency of his system. Here are some examples, where these principles can bring some benefits:

- If an agent makes requests for a given service, the agent who answers can change between two requests[1]. This contributes to increase the reliability of the multi-agent system. Indeed even if a skilled agent is removed, another could be found even if the designer had not explicitly anticipated it, or better, without need for the designer to anticipate it.
  We can imagine for example that the acquaintance architecture adapts to match the network performance architecture. Two agents a_1 and a_2 can provide the same service required by a client agent a_c. Depending on the localization of a_1 or a_2 in the network, or between any predefined acquaintance for a_c and one of the a_i, a_c will request only the provider whose answer is the fastest (without using a systematic broadcast !).
- If an agent performs the same task, he can "prefer" to learn a skill and thus remove the need to delegate its achievement.
  On the other side, if an agent is overwhelmed by requests from other agents who want to exploit one of his skills, he can choose to teach this skill to some other(s) agent(s) to multiply the supply and then lighten his burden.
- If for some reason an agent has to disappear from the multi-agent system and he owns some critical skill, he can teach it to some other agent and thus warrants the continuity of the whole multi-agent system.
- When the designer want to improve how a service is treated in its system, he can dynamically add a new agent with the new version of the skill and makes him teach it to the older-version-skilled agents to upgrade them.

To be able to apply these strategies, agents should be able to :

- dynamically create new acquaintance links in order to self adapt the organization ([9]). However they must first have a way to find the "right agent". Therefore a default message routing and default acquaintances must be provided for at least reaching the "right agent" through go-betweens.
- learn new skills from other agents (and therefore agents must be able to teach each other) (see [4]). A mechanism must be provided that supports it and the distributed aspect must be taken into account.

[1] Since the acquaintances are dynamically computed (according to some predefined rules of course).

- create new agents, and by using the learning/teaching ability, these agents could be tuned to what is needed.

Of course, agents will use these abilities autonomously and therefore behavioural strategies, for deciding when to apply them, must be created. There is a need to challenge some of the decisions from time to time. For example when a direct acquaintance link has been created, because at some time it was the most suitable, this may no longer be the case later and then a new adaptation is necessary. Thus, these strategies should integrate some mechanisms to call into question direct acquaintances that have been created.

3 Experiments

To experiment these principles, we need a framework that provides code mobility in order to apply the dynamic acquisition of skills. Thus, we used our multi-agent framework called MAGIQUE[2] [1, 11]. We will briefly introduce this framework and then experiment dynamic organizations of multi-agent systems with it.

3.1 MAGIQUE

MAGIQUE proposes both an organizational model [1], based on a default hierarchical organization, and an agent model [11], which is based on an incremental building of agents.

Dynamicity is a keypoint in MAGIQUE and the three principles of self-organization we have presented need this dynamicity in order to be implemented. The requirements made in section 2.3 are satisfied. We will insist on features that promote these aspects, other details will not be deepened here.

The agent model: building agents by making them skilled. The agent model is based on an incremental building of agents from an elementary (or atomic) agent through dynamical skill acquisition. A skill is a "coherent set of abilities". We use this term rather than service[3], but you can consider both as synonyms here. From a programmer oriented view, a skill can be seen as a software component that groups a coherent set of functionalities. The skills can then be built independently from any agent and reused in different contexts.

We assert that only two prerequisite skills are necessary and sufficient to the *atomic agent* to evolve and reach any wished agent: one to interact and another to acquire new skills [4].

Thus we can consider that all agents are at birth (or creation) similar (from a skill point of view): an empty shell with only the two above previously mentioned skills.

[2] Magique stands for the french "Multi-AGent hiérarchIQUE" which obviously means "hierarchical multi-agent".

[3] We keep *service* for "the result of the exploitation of a skill".

[4] Details can be found in [11].

Therefore differences between agents are issued from their "education", i.e. the skills they have acquired during their "existence". These skills can either have been given during agent creation by the programmer, or have been dynamically learned through interactions with other agents (now if we consider the programmer as an agent, the first case is included in the second one). This approach does not introduce any limitation to the abilities of an agent. Teaching skills to an agent is giving him the ability to play a particular role within the multi-agent system he belongs to.

For our purpose here, this ability to dynamically learn and teach skills is useful for the dynamic organization of the multi-agent system, in particular to make use of the second and third principles.

The organizational model. In MAGIQUE, there exists a basic default organizational structure which is a hierarchy. It offers the opportunity to have a default automatic mechanism to find a skill provider.

The hierarchy characterizes the basic structure of acquaintances in the multi-agent system and provides a default support for the routing of messages between agents. A hierarchical link denotes a communication channel between the implied agents. When two agents within a same structure are exchanging a message, by default it goes through the tree structure.

With only hierarchical communications, the organization would be too rigid and thus MAGIQUE offers the possibility to create direct links (i.e. outside the hierarchy structure) between agents. We call them *acquaintance links* (by opposition of the default *hierarchical links*). The decision to create such links depends on some agent policy. However the intended goal is the following: after some times, if some request for a skill occurs frequently between two agents, the agent can take the decision to dynamically create an acquaintance link for that skill. The aim is of course to promote the "natural" interactions between agents at the expense of the hierarchical ones.

With the default acquaintance structure, an automatic mechanism for the delegation of request between agents is provided. When an agent wants to exploit some skill, it does not matter if he knows it or not. In both cases the way he invokes the skill is the same. If the realization of a skill must be delegated to another, this is done automatically for him, even if he does not have a peculiar acquaintance for it. The principle of the skill provider search is the following:

- the agent knows the skill, he uses it directly
- if he does not, several cases can happen :
  - first he has a particular acquaintance for this skill, this acquaintance is used to achieve the skill (ie. to provide service) for him,
  - he has a team and someone in his subhierarchy knows the skill, then he forwards (recursively through the hierarchy) the realisation to the skilled agent,
  - he asks its supervisor to find for him some gifted agent and his supervisor applies the same delegation scheme.

One first advantage of this mechanism of skill achievement delegation is to increase the reliability of the multi-agent system: the particular agent who will perform the skill has no importance for the "caller", therefore he can change between two invocations

of the same skill (because the first has disappeared of the multi-agent system or is overloaded, or ...).

Another advantage appears at the programming stage. Since the search of a skilled agent is automatically achieved by the hierarchy, when a request for a skill is coded, there is no need to specify a particular agent. Consequently the same agent can be used in different contexts (i.e. different multi-agent applications) so long as an able agent (no matter which particular one) is present. A consequence is, that when designing a multi-agent system, the important point is not necessarily the agents themselves but their skills (i.e. their roles).

Obviously the evolutive default organizational structure with its automatic skill provider search offers the tools to apply the first of the previously mentioned principles.

The API These models have been put into concrete form as a JAVA API, called MAGIQUE too. It allows the development of multi-agent systems distributed over an heterogeneous network. Agents are developed from incremental (and dynamical if needed) skill plugging and multi-agent system are hierarchically organized. As described above, some tools to promote dynamicity in the multi-agent system is provided: direct acquaintance links can be created, new skills can be learned or exchanged between agents (with no prior hypothesis about where the bytecode is located, when needed it is exchanged between agents). The API is available and can be downloaded at `http://www.lifl.fr/MAGIQUE`.

3.2 Experiments

In this section we will present terse experiments that put into concrete form the principles of dynamic organization that have been described. These experiments have been completed with MAGIQUE[5].

The first experiment is concerned with the first principle : create the acquaintances that follow the natural flow of messages in the multi-agent system. The second deals with the second principle: the distribution of skills in the system is dynamically changed. The third experiment corresponds to the... third principle: new collaborators are created by an agent who wants to get rid of the need to treat too many requests for a given service.

First experiment: adapting the acquaintances organization This is a simple example where one agent, *SU*, is a service user and the needed service can be provided by two other agents, *SP1* and *SP2*. At the beginning, the multi-agent system is organized into a hierarchy and our three agents are located somewhere in the hierarchy but are not directly connected (cf. Figure 1). We do not show other agents since they do not interfere here. We have chosen to have *SP1* and *SP2* connected to the same root agent but this is of no importance nor influence. These agents are distributed over a network of workstations.

[5] The sources of these experiments can be downloaded at `http://www.lifl.fr/MAGIQUE/dynamicity`.

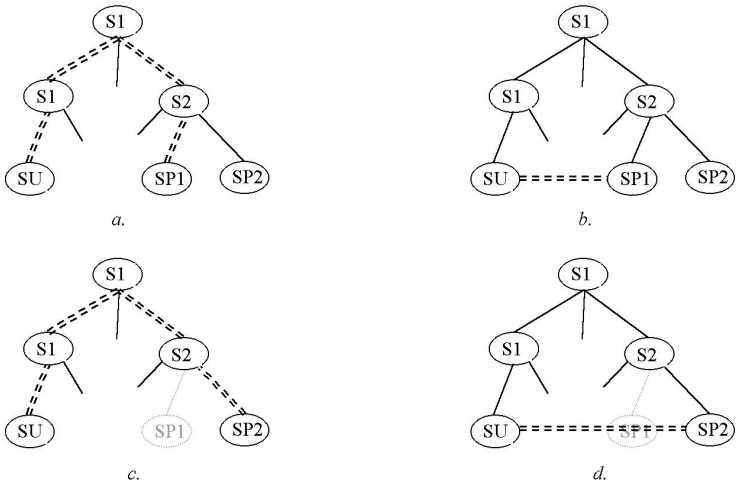

Fig. 1. Dynamic organization of acquaintances in a multi-agent system. *a.* Beginning: multi-agent system is hierarchically organized, service requests (see double dash lines) used the default hierarchical organization and *SP1* is reached. *b.* Self-organization: direct acquaintance link with *SP1* is created. *c. SP1* disappears: service requests use the default organization and *SP2* is reached. *d.* Self-organization: direct acquaintance link with *SP2* is created.

Agent *SU* sends at regular time requests for a service σ. Once the service has been performed, a payment request is sent back to *SU*, thus we have a way to measure the duration between the initial service request and the completion of the service.

At the beginning since *SU* does not know any skilled agent, the request is routed using the default hierarchical organization. According to the automatic skill provider search, *SP1* is reached (see Figure 1-*a.*).

After some requests, since the same *SP1* provides the service to *SU*, *SU* decides to create a direct acquaintance link with *SP1*. The decision is taken according to some criteria that can be chosen while the agent is designed (in this case a simple threshold decision process has been used). The direct link is now used (see Figure 1-*b.*) and as a consequence :

- the number of messages sent in the multi-agent system is reduced,
- agents *S1*, *S11*, *S12* are less “stressed” and can use their time to perform other tasks than routing messages,
- the delay before the service is finished is reduced.

Now, assume that agent *SP1* is removed from the multi-agent system. Then the default hierarchical organization is again used, and agent *SP2* is now reached (see Figure 1-*c.*). The direct benefit for the multi-agent system is fault tolerancy. Although an able agent disappears, the organization provides a way to find another able agent. This is automatically done for the service user, he performs the service requests in the same way as before.

Lastly, after some times the multi-agent system adapts again, and an acquaintance link between *SU* and *SP2* is created (see Figure 1-*d.*).

The tabular of figure 2 gives, for the 4 periods, the average durations between the moment a service σ request is sent and the moment the payment is achieved. The first line corresponds to a multi-agent system where only agents *SU*, *SP1* and *SP2* are working. In the second line, agents have been added to simulate load on *S*, *S1* and *S2*, and to generate extra network traffic. This is a more "realistic" situation. This explains differences between numbers in the first and third columns for the two rows.
Agents *SU*, *SP1* and *SP2* have been distributed over a network, and *SP2* was located in a different domain from the two others, this explains the slight difference between the results in columns two and four.

Fig 1-*a.*	Fig 1-*b.*	Fig 1-*c.*	Fig 1-*d.*
174.25	135.8	144.3	118.2
341.37	147.1	325.1	119.6

Fig. 2. Average durations in milliseconds before service achieved.

Second experiment: adapting the skill distribution This experiment is similar to the previous one. One agent, *SU*, is a service user and the required service can be provided by another agent *SP*. In the beginning, the multi-agent system is organized into a hierarchy and the two agents are located somewhere in the hierarchy (cf. Figure 3).

The scenario is the following: agent *SU* sends at regular time requests for a service σ. Once the service has been performed a payment request is sent back to *SU*, thus we have a way to measure the duration between initial service request and the completion of the service.

At the beginning since *SU* does not know any skilled agent, the requests is routed using the default hierarchical organization. According to the automatic skill provider search, *SP* is reached (see Figure 3-*a.*).

But after some times, according to some predefined policy of its own, *SU* decides to try to acquire from *SP* the skill that is required to achieve the service σ. If *SP* agrees, the skill is exchanged between agents (see Figure 3 -*b.*). No hypothesis has to be made about the location of bytecode for σ, it is physically exchanged between agents[6] if needed.

Of course, once *SU* has learned (or acquired), he is no more dependant on *SP* and service σ is satisfied faster (see Figure 3-*c.*). Moreover, *SP* do not need *SU* any more.

Now, if *SU* is disconnected from the system (see Figure 3 -*d.*), he can still achieve the service σ (or similarly if it is *SP* that leaves the system).

[6] More precisely, exchange is performed by the platform that hosts agents, since it is one of the principle of the implementation of the MAGIQUE API.

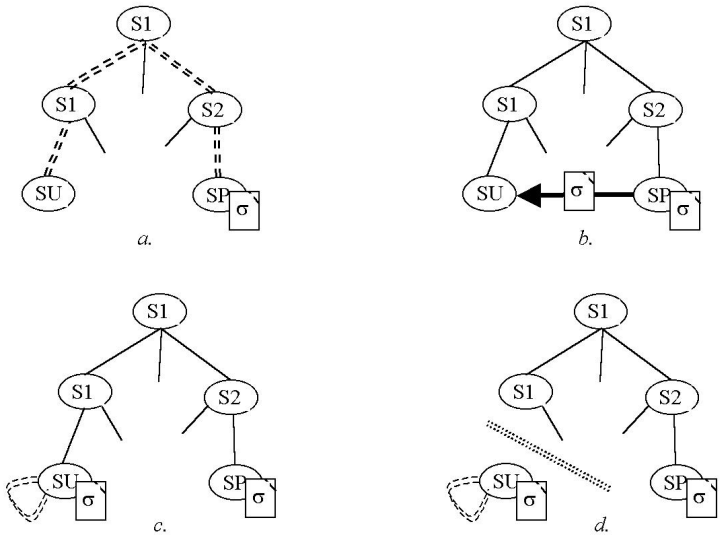

Fig. 3. Dynamic exchange of skill. *a.* Beginning: multi-agent system is hierarchically organized, service requests (see double dash lines) uses the default hierarchical organization and *SP* is reached. *b.* Exchange: skill σ is "learned" by *SU* from *SP*. *c. SU* use its "own" σ to achieve what he needs to. *d. SU* can even be disconnect from the system.

Giving figures like the previous experiment is not really meaningful. Before *SU* has acquired/learner the service, the time before the service σ is satisfied depends on how much *SP* and the hierarchy are loaded. After the skill acquisition, the delay to achieve the service for *SU* is reduced to the time needed to "run" it.

Third experiment: create a pool of apprentices In this experiment, an agent *SU* makes request to a service σ. This service can be provided by an agent *SP*. But *SP* is also the agent which provides some π service. This π service is highly requested by some π-user agents (see Figure 4-*a.*).

Therefore, *SP* is overwhelmed by requests to its π-skill and *SU*, who does not use π, suffers from that. To avoid this situation, *SP* creates a pool of agents to support him. He teaches to these agents the skill to achieve π and each time he receives a request for π, he spreads it to one of its apprentices (see Figure 4-*b.*). The consequence is of course, that *SP* can spend more time to satisfy other requests and in particular requests to σ. Thus, the global efficiency of the system is improved.

In this experiment, 8 π-users are used. They send n requests and simultaneously *SU* makes m requests for σ. Before the pool of apprentices is created (that is when *SP* is alone to satisfy all requests), the $n.\pi$ and $m.\sigma$ requests are all achieved after 52 seconds. When *SP* creates a pool of 3 agents, for the same $n.\pi$ and $m;\sigma$ requests, we obtain a time of 30.7 seconds. Of course, all these experimentations are just *proof of concept*, and particularly figures are given as examples.

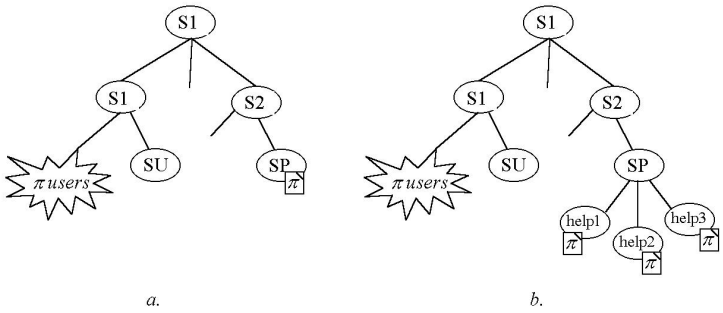

Fig. 4. Create pool of helpers. *a. SP* must satisfy request from π service users and from *SU*, he is overwhelmed by requests for π. *b. SP* has created 3 helper agents and teached them the π skill, he distributes requests for π to these helpers and thus lighten his burden.

4 Conclusion

Static organizations have defaults. In order to be efficient, there is a need to be reactive and to adapt the organization to the reality of agents exchanges. Our thesis in this paper is that the needs are the same for multi-agent systems. It is too difficult (and probably even impossible) for a multi-agent system designer to foresee the flow of messages within his system. Moreover, in some cases, there will not be only one designer but several that design only a piece of the system. It should be possible to rely upon generic strategies to manage dynamicity.

We have proposed some principles to adapt the organization in order to reduce the number of messages in the multi-agent system and to improve the delay before a request is satisfied: creation of new specific acquaintance relations to remove the go-between, exchange of skills between agents and creation of new agents to reduce overloading. Agents can apply these principles autonomously depending on some decision of their own. And the taken decision should be challenged after some times, to ensure that the current acquaintance is still the best choice.

Future works on this notion of dynamic organizations should be given a more formal frame, particularly by working and defining on an ontology that describe its semantic. Then, we could have agents that belong to several organizations, relying on different kinds of organizational models. But, they would be able to handle the dynamicity within those organizations.

References

[1] N.E. Bensaid and P. Mathieu. A hybrid and hierarchical multi-agent architecture model. In *Proceedings of PAAM'97*, pages 145–155, 1997.

[2] F. M. T. Brazier, B. M. Dunin-Keplicz, N. R. Jennings, and J. Treur. DESIRE: Modelling multi-agent systems in a compositional formal framework. *Int Journal of Cooperative Information Systems*, 6(1):67–94, 1997.

[3] Gero Vierke Christian Gerber, Jorg Siekmann. Holonic multi-agent systems. Technical report, DFKI GmbH, 1999.

[4] R.P. Clement. To buy or to contract out: Self-extending agents in multi-agent systems. In *SOMAS: A Workshop on Self Organisation in Multi Agent Systems*, 2000.

[5] Anand Rao David Kinny, Michael Georgeff. A methodology and modelling technique for systems of bdi agents. Technical report, Australian Artificial Intelligence Institute, 1996.

[6] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of ICMAS'98*, 1998.

[7] J. Ferber and O. Gutknecht. Operational semantics of a role-based agent architecture. In *Proceedings of ATAL'99*, jan 1999.

[8] Christian Gerber. Bottleneck analysis as a heursitic for self-adaptation in multi-agent societies. Technical report, DFKI GmbH, 1998.

[9] R. Ghanea-Hercock. Spontaneous group formation in multi-agent systems. In *SOMAS: A Workshop on Self Organisation in Multi Agent Systems*, 2000.

[10] E. A. Kendall, M. T. Malkoun, and C. H. Jiang. A methodology for developing agent based systems. In Chengqi Zhang and Dickson Lukose, editors, *First Australian Workshop on Distributed Artificial Intelligence*, Canberra, Australia, 1995.

[11] JC. Routier, P. Mathieu, and Y. Secq. Dynamic skill learning: A support to agent evolution. In *Proceedings of the AISB'01 Symposium on Adaptive Agents and Multi-Agent Systems*, pages 25–32, 2001.

[12] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. In *Proceedings of the 1st ICDCS*, pages 186–192. IEEE Computer Society, 1979.

[13] M. Wooldridge, NR. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 2000.

[14] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Organisational rules as an abstraction for the analysis and design of multi-agent systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):303–328, 2001.