Philippe Mathieu, Jean-Christophe Routier et Yann Secq
Laboratoire d'Informatique Fondamentale de Lille
Cité Scientifique 59655 Villeneuve d'Ascq Cedex
{mathieu,routier,secq}@lifl.fr

Abstract.[1]

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[**?**] methodology, for instance, identifies this stage as the *acquaintance model*, which is defined as :

> *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 $a \rightarrow b$ indicates that a will send 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 [**?**], it is stated that :

[1] Article publié dans *Proceedings of PRIMA 2002, Tokyo, LNAI 2413. ISBN 3-540-44056-7. pp 109-122. August 2002.*

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 the fact that agents interact and so need to have some communication paths to exchange information. Other methodologies [**?**, **?**], often state the same kind of concepts but seldom identify that the acquaintance structure is a first-class citizen of MAS.

It is true that some works highlight the importance of the notion of organization in multi-agent systems : the Contract-Net Protocol [**?**] is based on a market-type system, the Aalaadin [**?**] model relies on the idea that agents are identified by the roles they hold within some groups, the Magique [**?**] model proposes a hierarchical structure and, lastly, the holonic approach [**?**]. Unfortunately, these works seldom reify this notion.

Moreover building an organization to optimize agent interactions is not straightforward : how should we spread funtionalities among agents, how is it possible to reduce the cost of communication, and overall how can the system deal with agents that freely leave or join it? Lastly, how can organizations deal with the ever-changing flow of agent interactions?

This paper postulates that this complexity should not be exclusively addressed by the multi-agent system designer. The infrastrcuture of organizations infrastructures should provide default behaviours to dynamically optimize communication flow, in order to lower the number of messages that are exchanged, or to improve the quality of service. Too little works have been done in this direction [**?**].

Thus, we propose three rather simple and natural principles inspired from social organizations, that can be applied to adapt multi-agent organizations. These principles lead to a modification of the acquaintance organization and to a dynamic modification of the skills of the agents or even to the creation of new agents. This implies a modification of the distribution of the roles and of the dependences network between agents. Firstly the acquaintance structure tends to map to the dependence structure with the creation of new direct communication channels, secondly the distribution of the skills among the agents in the system is dynamically changed. Moreover we want this ability to self adapt to be a primitive feature of the system and not to be chargeable to the multi-agent system designer or dependent upon the agent model.

In 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 these problems before we apply their solutions to multi-agent systems. The second section illustrates the dynamic organizations through some simple experiments performed with the Magique framework which will briefly be introduced.

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 problems, 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 an enough 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 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 sys tem be organizedin such a way that no agent becomes a critical overloaded resource[**?**]? 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. But when it is possible to avoid it, it would be better if one agent does not become a bottleneck in the system 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 can we ensure that one of the service provider agents will not be overused and others ignored.

Lastly, we will consider a situation where we consider the "client of service" point of view rather than the service provider one. One agent may have to often 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 others. Agents must adapt themselves to such a dynamic environment. The designer can not predetermine these 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 does 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 he/she/it is. But, of course, this may be a source of problems. You do not necessarily want to use some middleman that can know what you want from the "right person" and then make use of this information for his personal advantage. Moreover, this can have a cost since the middleman can ask for a payment only for having helped you to get in touch with the right person. Therefore after a time, when you have obtained the information you needed, you can try to reach the right person directly. This implies that you communicate directly with the person you effectively depend on.

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

If you consider the client's point of view, making too frequent requests to some critical resource is a major drawback which has a cost. Either a time cost because clients 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 circumvent 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 problem mentioned earlier. It may seem that this principle could have been called "remove the middleman", however this must be moderated. Indeed, creating new (social) links has a cost and it is not always appropriate to circumvent the middleman. He may know his job, and his offer for a given service can change because he has had found of a more able provider. In such a case the use of the middleman would have been beneficial. In consequence, "having a good address book" does not always mean removing the middleman, 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 intent is that, after its training, the apprentice becomes a new resource. Of course, 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 acquaintance 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 among the agents. The idea is to have an evolution of the structure of acquaintances where the dependence links are favoured at the expense of predefined ones.

Of course the major benefit should be for the designer of the multi-agent system who can prepare his system as he sees most suitable and then rely on these principles to adapt the efficiency of his system. Here are some examples, where these principles can be of considerable benefit:

- Applying the first principle, the dependence network tends to coincide with the acquaintance network.
- By learning new skills, and agent increases its autonomy.
- If an agent makes requests for a given service, the agent who answers may not be the same one between two requests[2]. This contributes towards increasing 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 acquaintances for a_c and one of the a_i, a_c will request only to the provider whose answer is the fastest (of course without using a systematic general broadcast).

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

- If an agent performs the same task, he can "prefer" to learn a skill and thus remove the need to delegate in order to perform the task.

  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 agent(s) to multiply the offer 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 guarantee the continuity of the whole multi-agent system.

  This has some similarities with the work in [**?**] where the *Adapative Agent Architecture* is proposed: the author use dynamic re-organization with middle-agent (or broker) to improve robustness and promote fault-tolerance. However our goal concerning self-organization is more general since we provide dynamic self organization in order to improve the interactions between agents according to their natural effective dependences (even if this sometimes means to remove this dependence when skills are exchanged, as we will see later).

- When the designer wants to improve how a service is treated in his system, he can dynamically add a new agent with the new version of the skill and make him teach it the other older-version-skilled agents to upgrade them.

To be able to practically apply these strategies, some abilites must be provided by the framework, agents should be able to :

- dynamically create new acquaintance links in order to self adapt the organization ([**?**]) to match the dependence links. 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 middle-agents.
- learn new skills from other agents (and therefore agents must be able to teach each other) (see [**?**]). A mechanism must be provided that supports it and the distribution aspect must be taken into account.
- 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 the 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[3] [**?**, **?**]. 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 [**?**], based on a default hierarchical organization, and a mimimal agent model [**?**], which is based on an incremental building of agents. Magique is dedicated to the *implementation* of multi-agent systems. Magique is not a multi-agent applications but a support for such applications. Thus, it does not directly provide high level features like knowledge base management, planners, etc.

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 taken into consideration here.

3.1.1 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[4], 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 re-used in different contexts.

We assert that only two prerequisite skills are necessary and sufficient to the *atomic agent* to evolve and reach any desired agent: one to interact and another to acquire new skills (details can be found in [**?**]).

These skills are indeed *necessary*. Without the "skill acquisition" skill, such an agent is just an empty shell unable to perform any task. Without the interaction skill, an agent is isolated from the "rest of the world" and therefore loses any interest. Moreover without communication an agent will not be able to learn new skills from others.

They are *sufficient* since it suffices to an agent to use his interactive skill to get in touch with a gifted agent and then to use his acquirement skill to learn some new talent. Then every ability can be given to an agent through learning from a "teacher". Let us precise that the exchanged skills are "stateless", it is the functional ability that is exchanged not some kind of experience.

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-mentioned skills.

We claim that every model of agent proposed by the various existing definitions [**?**] matches this definition. Indeed, it "suffices" to build the skills that provide the basic abilities of the model and to teach them to an atomic agent.

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

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

Thus, we do not want to use a particular agent model, the "high level intelligent" abilities can be chosen and evolve at will.

Therefore, differences between agents issue 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.

3.1.2 The organizational model.

Throughout the following the agents are like these described in the previous section and are supposed to be co-operative agent and not self-interested one.

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 the 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 *dependence 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 a dependence link for that skill. In fact, to be more precise, we must say that an acquaintance link corresponding to a dependence link is created. The aim is of course to promote the "natural" interactions between agents at the expense of the default hierarchical ones.

With the default acquaintance structure, an automatic mechanism for the delegation of requests between agents is provided without the use of a general costly broadcast. 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 skills 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 particular 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 occur
  - if he has a particular acquaintance for this skill, this acquaintance is used to achieve the skill (ie. to provide service) for him,

- else, he has a team and someone in his sub-hierarchy knows the skill, then he forwards (recursively through the sub-hierarchy) the realisation to the skilled agent,
- else, he asks his supervisor to find for him some competent agent and his supervisor applies the same delegation scheme.

In this mechanism, some agents play the role of middle-agents as defined in [**?**]: "*Agents (...) that are* neither *requesters* nor *providers*". But let us precise, that this is just a temporary state: these agents are not dedicated to be exclusively middle-agents and can send requests or provide services at other moments.

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 from 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 organization, 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 (ie. their roles).

Obviously the evolutive default organizational structure with its automatic skill provider search offers the tools to apply the above-mentioned principles.

3.1.3 The API

These models have been put into concrete form as a JAVA API. It allows the development of multi-agent systems distributed over heterogeneous network. Agents are developed from incremental skill plugging (and dynamically if needed) and multi-agent system are hierarchically organized. As described above, some tools to promote dynamicity in the multi-agent system are provided: direct communication links can be created, new skills can be learned or exchanged between agents (with no prior hypothesis about where the byte-code is located, when needed it is exchanged between agents). This API can be downloaded at `http://www.lifl.fr/SMAC` topic Magique. We have used it to developped a co-operative work application [**?**] or a framework for distributed calculus [**?**].

3.2 Three experiments for three principles.

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

[5]The source codes of these experiments can be downloaded at `http://www.lifl.fr/MAGIQUE/dynamicity` and experiments can then be reproduced.

The first consists in creating the acquaintances that suit the best to the natural flow of messages in the multi-agent system. In the second, the distribution of skills in the system is dynamically changed. While in the third new collaborators are created by an agent who wants to get rid of the need to treat too many requests for a given service.

3.2.1 First experiment: adapting the acquaintances organization

This is a simple example where one agent, *SU*, is a service user and the required service can be provided by two other agents of the multi-agent system, *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 in the multi-agent system 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 or influence. These agents are distributed over a network of workstations.

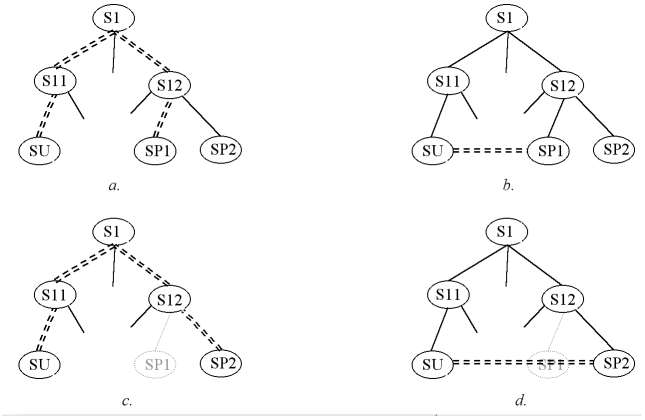

Figure 1: *Dynamic organization of acquaintances to match dependence links. a.* Beginning: multi-agent system is hierarchically organized, service requests (see double dash lines) use the default hierarchical organization and *SP1* is reached. *b.* Self-organization: direct communication link that corresponds to a concrete dependence with *SP1* is created. *c. SP1* disappears: service requests use the default organization and *SP2* is reached. *d.* Self-organization: direct communication link with *SP2* is created.

Agent *SU* sends at regular intervals 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 time which has elapsed between the initial service request and the completion of the service (the same criterium is used in [**?**]).

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, *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 communication link with *SP1* in order to favour the dependence. The decision is taken according to some criteria that can be customized 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 consequences:

- the number of messages sent in the multi-agent system is reduced,
- the agents *S1*, *S11*, *S12* are less "stressed" and can use their time to perform other tasks than routing messages and being used as middle-agents,
- thirdly 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 tolerance. 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 table at figure 2 gives, for the 4 periods, the average time 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.

When the direct communication link is created, the middle-agents *SI*, *SI1* and *SI2* are no more used. We can see the performance enhancement of it in the differences between columns *a* and *b*.

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

Figure 2: Average durations in milliseconds before service achievement

3.2.2 Second experiment: adapt 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*.

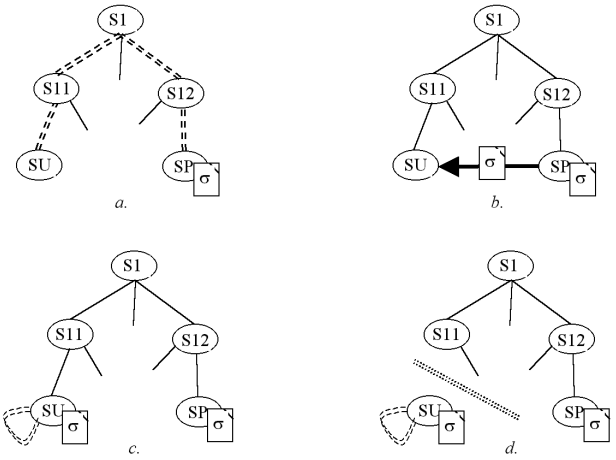

Figure 3: *Dynamic acquisition of skill. a.* Beginning: multi-agent system is hierarchically organized, service requests (see double dash lines) use the default hierarchical organization and *SP* is reached. *b.* Exchange: skill σ is "learned" by *SU* from *SP*. *c. SU* uses its "own" σ to achieve what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

At the beginning since *SU* does not know any skilled agent, the requests are 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 his own, *SU* decides to acquire from *SP* the skill that is required to achieve the service σ. Since *SP* agrees, the skill σ 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, to avoid confusion with the "learn" term) the skill, he is no longer dependant on *SP* and service σ is satisfied faster (see Figure 3-*c.*). Moreover, *SP* is freeded from the need to "help" *SU*. *SU* has increased his automony.

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

Giving figures like the previous experiment is nor really meaningful. Before *SU* has acquired/learned the service, the time before service is carried out depends on how much *SP* and the hierarchy are loaded. After the skill acquisition, the time elapsed to perform the service for *SU* is reduced to the time needed to invoke it locally and disconnection of agent *SP* or *SU* is of no consequence on the achievement of σ.

3.2.3 Third experiment: create a pool of apprentices

In this experiment, an agent *SU* makes requests to a service σ. This service can be provided by an agent *SP*. But *SP* is also the agent which provides some π

[6]More precisely, exchange is performed by the platforms that host the agents, since it is the principle of the implementation of the Magique API.

service. We assume this π service to be highly requested by some π-user agents (see Figure 4-*a.*).

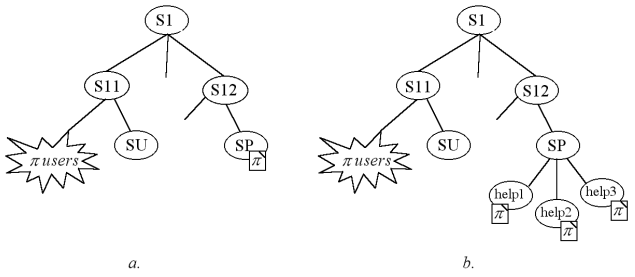

Figure 4: *Create pool of apprentices. a. SP* must satisfy requests from π service users and from *SU*, he is overwhelmed by requests for π. *b. SP* has created 3 apprentice agents and taught them the π skill, he distributes requests for π to these apprentices and thus lightens his burden.

Therefore, *SP* is overwhelmed with 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 these agents the skill required to perform π and each time he receives a request for π, he dispatches it to one of his apprentices (see Figure 4-*b.*). The consequence is of course, that *SP* can spend more time satisfying 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 $n.\pi$ et $m.\sigma$ 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 experimentats are just *proofs of concept*, and in particular figures are given only as examples.

4 Conclusion

Static organizations have drawbacks. In order to be efficient, there is a need to be reactive and to adapt the organization to the reality of the exchanges. Our theme 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 (and moreover for a team of designer) to foresee the flow of messages within his system. It should be possible to rely upon generic strategies to manage dynamicity of exchanges.

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 middle-agents,
- exchange of skills between agents to increase autonomy,

- creation of new agents to reduce overloading.

A consequence of the application of these principles is a modification of the dependence network. 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. Our position is that such abilities must be provided as basics in a multi-agent framework.

Future works on this notion of dynamic organizations should be given a more formal framework, particularly by working on and defining an ontology that describes 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 these organizations.