Dynamic Organization of Multi-Agent Systems

P. Mathieu  
Laboratoire d’Informatique  
Fondamentale de Lille  
Cité Scientifique 59655  
Villeneuve d’Ascq Cedex  
mathieu@lifl.fr

J.C. Routier  
Laboratoire d’Informatique  
Fondamentale de Lille  
Cité Scientifique 59655  
Villeneuve d’Ascq Cedex  
routier@lifl.fr

Y. Secq  
Laboratoire d’Informatique  
Fondamentale de Lille  
Cité Scientifique 59655  
Villeneuve d’Ascq Cedex  
secq@lifl.fr

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 dynamic evolution of organizations are seldom provided in current framework propositions.

In this paper we discuss about self-adaptation of organizations in multi-agent systems according to the dynamic of interactions between agents. Starting from a default organization, architecture of acquaintances evolves autonomously depending on message flow in order to improve the global behavior 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 others 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 others agents, but they seldom provide guidelines to design the acquaintances structure. The GAIA[12] methodology, for instance, identify 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 $a \rightarrow b$ 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, 9], often state the same kind of concepts but seldom identify that the acquaintance structure is a first-class citizen of MAS entities.

Anyway, some works highlight the importance of the notion of organization in multi-agent systems. Unfortunately, those works seldom reify this notion. For example, the Contact-Net Protocol [11], 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 [6] 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 [10] 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 claims that this complexity should not be exclusively addressed by the multi-agent system designer. Organizations infrastructures should provide default behaviors 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 [8, 4] have been done in this direction.

In first section, we describe the needs to have an adaptative 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 use it to illustrate dynamic organizations through some simple experiments described in section three.

2. ADAPTING THE ARCHITECTURE OF THE ORGANIZATION

Before we have a look on 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 first problem, and probably the basic one, is how acquaintances are created? We mean, how an agent can have information about the existence of another able agent. One solution is of course that this can be predetermined and established by the multi-agent system designer, but this is not an enough satisfactory answer. First, how should this designer proceed to choose the most fitted acquaintance architecture, which methodology must be applied, if there exists any really convenient. And second, 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 to organize the system in a way such that no agent becomes a critical overloaded resource[7]? This implies that even if an organizational structure has been chosen, this is not enough you need to chose how the skills are distributed among the agents? It is of course difficult, if not impossible, to give universal rules to do this. If it is possible to avoid it, it would be better if one agent could 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 to predetermine it? 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 service provider agents will not be overused and others ignored.

Last, 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 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 much requests, at least because of the communications. It would have been better, while designing the system, to qualify this agent with the service, or to allow the agent to dynamically acquires 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 another time. Agents must adapt themselves to this dynamic environment. The designer can not predetermine those situations. Therefore the only thing he can do is to prepare his agents such that they can adapt autonomously to the changes that occurs 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 those problems?

The problems we have brought out 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 “good person for the job” appears. Often this “good 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 source of problems. You do not necessarily want to use some go-between that can know what you want from the “good person” and make use of this information. Moreover, this can have a cost since the middleman can ask for a payment only to have help you to get in touch with the good person. Therefore after some time, when information has been gathered you try to reach the good person directly.

The problem of overloaded resources exists too. The more a person or a society is capable, the more it is probable 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 increases. 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 requests to some critical resource too often 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 client must pay for the service. Therefore, when it is possible, clients must try to go round this dependence.

In this three cases, the problem of cost or efficiency appears. In social organizations, there is a trend to reach better efficiency. This trend can be natural – we all have tendency to apply the least effort law –, 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 behavior and 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 of course with the first problem mentioned earlier. 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. Moreover this one can know his job well and his offer for a given service can change because he has had knowledge of a more able provider. In such case the use of the go-between would have been beneficial. In consequence,
“having a good address book” does not mean to always remove the go-between, but rather to know when to use him and when not.

Second and third principles are rather means to tackle second and third problems and more generally to improve efficiency by reducing the delay before a request for a service can be satisfied. When a service company is overused, in order not to lose client, it will probably recruit able collaborators. The same is true, when the company needs a new skill, it can recruit new collaborators with the wanted competence. Or consider a craftsman with too much demands, 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 cannot 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, first, the number of messages exchanged in the system and, second, the delay before a service request can 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. And the idea is to have an evolution of the structure of acquaintances where the natural links are favored 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 multi-agent system as it seems the most well 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, since even if a skilled agent is removed, another could be found even if the designer had not explicitly anticipated it, or rather, 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 that needs a client agent \(a_c\) and depending on the localization of \(a_1\) or \(a_2\) in the network or any predefined acquaintances for \(a_c\) with one of the \(a_i\), after some times \(a_c\) will make request only to the one for which the answer of the network is the faster (of course without using a systematic general 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 chose to teach this skill to some other(s) 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 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 the other older-version-skilled agents to upgrade them.

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

- dynamically create new acquaintance links in order to self adapt the organization ([8]). However they must first have a way to find the “good agent”. Therefore a default message routing and default acquaintances must be provided for at least reaching the “good agent” through go-betweens.

- learn new skills from other agents (and therefore agents must be able to teach other) (see [4]). A mechanism must be provided that support it and the distributed 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 behavioral strategies, for deciding when to apply them, must be created. There is need too 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 fitted, this can no more be the case later and then new adaptation is necessary. Thus, those strategies should integrate some mechanisms to call into question those created direct acquaintances.

3. EXPERIMENTS

We have applied those principles using our multi-agent framework called MAGIQUE\(^2\) [1, 10]. 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 [10], 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 are strongly embedded in MAGIQUE: 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.

\(^1\)Since the acquaintances are dynamically computed (according to some defined rules of course).

\(^2\)Magique stands for the french “Multi-AGent hiérarchIQUE” which obviously means “hierarchical multi-agent”.
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\(^3\), but you can consider both as synonyms here. From
a more 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 suffi-
cient to the atomic agent to evolve and reach any wished agent: one
to interact and one 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.

Therefore differences between agents are issued from their “edu-
cation”, 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 inter-
actions 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 limitations to the abilities of an agent. Teaching
skills to an agent is giving him the possibility to play a particu-
lar role into 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 sys-
tem, in particular to make use of the second and third principles.

3.1.2 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 rout-
ing of messages between agents. A hierarchical link denotes a com-
munication channel between the implied agents. When two agents
of a same structure are exchanging a message, by default it goes
through the tree structure.

With only hierarchical communication, the organization would be
too rigid and 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 skills is the same. If the
realization of a skill must be delegate to another, this is done trans-
parently 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 ac-
    quaintance is used to achieve the skill (ie. to provide
    service) for him,
  - he is a supervisor and someone in his hierarchy knows the
    skill, then he forwards (recursively through the hierar-
    chy) 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 dele-
gation 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 had 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 programmed, there is no need to speci-
ify 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 conse-
quence 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 au-
tomatic skill provider search offers the tools to apply the first of the
previously mentioned principles.

3.1.3 The API
These models have been put into concrete form as a JAVA API,
called MAGIQUE10. It allows to develop multi-agent systems dis-
tributed over 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 are
provided: direct acquaintance links can be created, new skills can
be learned or exchanged between agents (with no prior hypothe-
sis about where the bytecode is located, when needed it is trans-
ferred between agents). The API can be downloaded at http:
//www.lifl.fr/MAGIQUE.

3.2 Experiments
In this section we will present small experiments that put into con-
crete form the principles of dynamic organization that have been
described. These experiments have been completed with MAG-
IQUE\(^5\).

First experiment is concerned with first principle : create the ac-
quaintances that fit the best in the natural flow of messages in the

\(^3\) We keep service for “the result of the exploitation of a skill”.
\(^4\) Details can be found in [10].
\(^5\) The sources of these experiments can be downloaded at http://
www.lifl.fr/MAGIQUE/dynamicity and experiments can then be reproduced.
multi-agent system. Second deals with the second principle: the distribution of skills in the system is dynamically changed. 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.

3.2.1 First experiment: adapt the acquaintances organization

It is a simple example where one agent, SU, is a service user and the needed service can be provided by two others 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 nor influence. The agents are distributed over a network.

![Figure 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.](image)

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 moment the service can be considered finished.

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, SP1 is reached (see Figure 1-a.).

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

Then, it occurs that agent SP1 is removed from the multi-agent system. Therefore the default hierarchical organization is again used, and this time agent SP2 is 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, when any. This is transparently done for the service user who has nothing special to do, he performs the service requests in the same way as before.

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

The tabular at 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. First line corresponds to the case where only agents SU, SP1 and SP2 are working. In 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 the differences between numbers in the first and third columns for the two rows.

Agents SU, SP1 and SP2 had 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.

```
<table>
<thead>
<tr>
<th>Fig 1-a</th>
<th>Fig 1-b</th>
<th>Fig 1-c</th>
<th>Fig 1-d</th>
</tr>
</thead>
<tbody>
<tr>
<td>174.25</td>
<td>135.8</td>
<td>144.3</td>
<td>118.2</td>
</tr>
<tr>
<td>341.37</td>
<td>147.1</td>
<td>325.1</td>
<td>119.6</td>
</tr>
</tbody>
</table>
```

Figure 2: Average durations in milliseconds before service is fully done

3.2.2 Second experiment: adapt the skill distribution

This experiment is as simple as previous one. One agent, SU, is a service user and the needed service can be provided by another agent SP. In the beginning, the 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 moment the service can be considered finished.

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 declared policy of its own, SU decides to try to acquire from SP the skill that is required to achieve the service σ. SP agrees therefore, skill is exchanged between agent (see Figure 3-b.). No hypothesis has to be made about the location of bytecode for σ, it is physically exchanged between the agents6 if needed.

6More precisely, exchange is performed by the platform that hosts the agent, since it is the principle of the implementation of the MAGIQUE API, but technical details in addition will not change understanding here.
Of course, once SU has learned (or acquired, to avoid confusion with “learn” term), he is no more dependant on SP and service σ is satisfied faster (see Figure 3-c). Moreover, SP is rid of the need to “help” SU.

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

Giving a tabular like the one for the previous experiment is of little interest. Before SU has acquired/learner the service, the delay before service 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 and disconnection is of no consequence.

3.2.3 Third experiment: create a pool of apprentices

In this experiment, an agent SU make 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 in demand 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 these 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 the experiment, that can be downloaded and reproduced too, 8 π-users are used. They send n requests and simultaneously SU makes m requests for σ. Before the apprentices pool is created (that is SP is alone to satisfy all the requests), the n π et m σ requests are all achieved after 52 seconds. Then SP creates a pool of 3 agents, for the same n π and m σ requests, we obtain a time of 30.7 seconds.

Figure 3: Dynamic exchange of skill. a. Beginning: multi-agent system is hierarchically organized, service requests (see double dash lines) used 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 remainder of the system.

Of course, all those experimentation are just proof of concept, and particularly figures are given as examples.

4. CONCLUSION

Static organizations have defaults. In order to be efficient, there is need to be reactive and to adapt to the reality of the exchanges in the organization. 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 in his system. Moreover, in some cases, there will not be only one designer but several (possibly a lot of) that design part of the systems and agents. It should be possible to rely upon 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 some overload. 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 provider.

Future works on this notion of dynamic organizations should be given a more formal frame, particularly by working and defining on ontology to 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.

5. REFERENCES


