Dynamic Skills Learning : a Support to Agent Evolution

JC. Routier ; P. Mathieu ; Y. Secq
Laboratoire d'Informatique Fondamentale de Lille
Cité Scientifique 59655 Villeneuve d'Ascq Cedex
{routier,mathieu,secq}@lifl.fr

Abstract

In this paper, we show that every agent can be built from an atomic agent through dynamic *skill* acquisition, a skill being a coherent set of abilities. At first, we propose a definition of an *atomic agent* and then we present the skill notion. In our work, since an agent is defined by the set of roles he can play according to the skills he has learned, we can consider that skills are the backbone of agents. We propose that the learning be dynamic and thus agents can effectively evolve during their "life". That means that the roles he plays can change. This approach promotes evolutivity, reusability and modularity.
These concepts have been applied to our own multi-agent system model, called MAGIQUE. It is based on hierarchies of agents (or agents recursively built from agents depending on the vision). This organisation provides an automatic delegation of the exploitation of skills between agents, that contributes to the simplicity and adaptability of MAS building.
An API that implements these ideas has been developed and of the corresponding API that. It provides an easy to use framework to build multi-agent applications where agents effectively dynamically evolve.

1 Introduction

The agenthood in an application is often hidden inside the programmer's mind (Shoham (1993); Travers (1996)). This has contributed to the rise of various definitions of the *agent* term (see Franklin and Grasser (1996)), and it is not a difficult task to collect a great amount of different definitions. The intersection between all these definitions is often not empty but they differ, sometimes slightly, by some functionalities that are basic for each model.

Starting from this observation, we want to try to propose a basis above which every other definitions could be settled. Our purpose is to define an *atomic agent* which can evolve dynamically in different ways in order to match the different notions that are introduced.

Our central point is the notion of *skill*. It denotes a coherent set of abilities that can be "given" to an agent. The principle is to start from an atomic agent and then to *teach* him some skills in order to build the wanted agent. The "result" depends on the teached skills and therefore different kinds of agents can be reached. Moreover the teaching can also be done during the agent activity and thus an agent can evolve dynamically, that means that the role he plays changes.

From a programming point of view, this approach promotes reusability and modularity because once a skill has been developed, it can be used many times in different contexts. Indeed, in this vision, a skill can be considered as a software component (for one other approach building agents from components see Horling and Lesser (1998) for example).

We claim that the different concepts for agents can be developped from that basis. As an illustration we have applied it to our own multi-agent model named MAGIQUE [1] (Bensaid and Mathieu (1995, 1997)). It is based on a hierarchic (or recursive) vision of a multi-agent systems. MAGIQUE is a multi-agent framework. We insist on the term *framework*: MAGIQUE is not a multi-agent system (MAS) but a support to build MAS. And as this, it allows the programmer to define his own policies for agent and MAS building since MAGIQUE gives to him the primitives to do it. In particular, MAGIQUE is based on the previously mentioned principle of building agent through skill acquisition. And then MAGIQUE provides the tools required to develop multi-agent applications where the agents dynamically evolve through skill acquisition. How the learning is done is the application part. Consequently, the ideas that issues from learning/acquisition works can be effectively put into concrete form with MAGIQUE.

Before we continue, a precision must be made. Throughout the following we are going to use the term "skill learning". This term could lead to confusion. Maybe the terms "skill acquisition" or "skill exchange" would be more fitted. We do not mean with the term "skill learning" that the agent learns from observation or some convergence algorithm. Rather, we mean that the agent has acquire a new skill that has been given to him by another agent (called the teacher, this can or not lose the skill when he gives it).

Thus, we present first a model, where the agents are dynamically built by dynamic skill acquisition and second an application framework, where multi-agent appli-

[1] Magique stands for the french "Multi-AGent hiérarchIQUE" that (obviously) means... "hierarchical multi-agent".

cations can be built according to this model. This provides the tools to perform the dynamic skill exchange and acquisition between agent. Let us note, that since it is a framework, the application part, such as the policies used to decide when a skill must be acquire or not, is left to the user.

The first section presents definitions: skills and agents. In the second one we give a brief description of our model MAGIQUE. The third should give an idea of the corresponding API that provides an easy to use framework to build applications where agents effectively dynamically evolve. Last we write some words about an application example where dynamic evolution of roles through skill is used.

2 Agent from Atomic Agent

2.1 Definitions

2.1.1 Skills

An agent is *one who acts*. In order to perform some activity, he must have the *skill* to do it. We define a skill as follows:

Definition 1 A *Skill* is a coherent set of abilities.

Then, a skill is a set of functionalities that can be exploited by an agent. We want everything being defined in term of skills, even the way an agent manages his skills. One could speak of meta-skills in this case, and even these should be able to dynamically evolve as explained before.

From a more pragmatic view, a skill should be seen as a software component whose public interface constitutes the abilities a capable agent can use.

The granularity and degree of complexity of a skill can not be definitively stated. The ability to parse an XML message or the ability to add two integers can each represent a skill although their complexities will probably be considered as being of different levels.

Moreover, whether you must group the four basic operations (addition, subtraction, multiplication and division) of integers in one skill or separate them in four skills, can not be definitively established. However it should be possible to reach a general agreement by saying that XML parsing and addition should be set in two different skills. That is what the "*coherent*" means in the previous definition.

Stated that to a skill should correspond only one ability (or conversely) could seem reasonable, but answer is not so easy, and in any case this will probably not withstand the confrontation with the reality of programmers... The problems that arise here are some common ones encountered in software engineering, and in OOP in particular, concerning (object) decomposition.

2.1.2 Agents

Now, the following definition should be able to receive a rather general agreement:

Definition 2 An *Agent* is an entity gifted of skills.

Any property commonly linked to the agent notion – such as proactivity, interactivity, intelligence, etc. – can indeed be expressed in term of *skills*. Therefore, it seems reasonable to say that all agent definitions, as those cited in Franklin and Grasser (1996), can be obtained from this one: the differences between two given agent definitions come from the basic functionalities assigned to the agents, that is from their skills.

However, for that reason, this definition can also be taxed as being too nebulous since it allows many interpretations depending on the skills attached to the agent. Thus we are going to precise it by stating a minimal set of skills.

We assert that only two prerequisite skills are necessary and sufficient to define an *atomic agent* from which every other agent definition can be established. These skills are: first, one skill that allows the agent to acquire new skills, and second, one skill for communications (with other agents – who could be human or software agents).

These skills are indeed *necessary*. Without the "skill acquirement" 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 skill from others.

They are *sufficient* too 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 acquire some new talent. Then every aptitude can be given to an agent through learning from a "teacher".

Consequently we propose the following new agent definition:

Definition 3 An *atomic agent* is an entity gifted of two skills: one to interact and one to acquire new skills. An *agent* is an atomic agent who has acquired skills from communications.

We claim that every agent proposed by the various existing definitions match this definition. Thus an agent using KQML is an agent gifted of such a "KQML understanding" skill; another who can encrypt his messages is an agent gifted of a code/decode skill (whatever algorithm is used), etc. At a more conceptual level, the notions of role and group, that are the core of the Aalaadin model Ferber and Gutknecht (1998), can also be translated in term of skills and this model could then be described in such terms (you must have skills to join or leave a group, to communicate inside a group, etc.).

Let us note that this is not the skills themselves that are important but rather their functionalities. Thus, we

can imagine that the interactive skill used by an agent can change during his life cycle, because he learns a new one for example. The important point is not that the agent have a particular interactive skill, but that the agent always has the ability to communicate (and similarly for the "learning" skill).

With no intention to enter into a philosophical debate, we can meanwhile note that this definition applies to the "human agent" who, since he can communicate and is able learn new knowledge, can evolve step by step through interactions with others (human and environment) agents. And communication (through only the five senses at the beginning and, after some times of evolution, with the contribution of the language later) and learning abilities seems to be effectively its sole attributes at the beginning of his life, and they allows a complete and variable education. Moreover, people who suffers from communication or learning troubles have difficulties to join the community (the "human MAS").

2.2 Agent Education

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

Therefore the 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[2].

2.3 Advantages

This paradigm, dynamic construction of agent from skills, has several advantages from a programming point of view.

development becomes easier Building an agent is teaching him some skills. Then agent programming is "reduced" to skill programming, but once a skill have been developed it can be used in different contexts.

Skills can be seen as *software components*, with all the advantages linked to this notion : modularity, reusability, etc.

efficiency An agent can decide to delegate the achievement of some task to another one (if this one accepts it). But this has a cost (due to communication for example) and is dependent upon the "good will" of the other. That's why in some case, if he has to often perform the same task, an agent can "prefer" to learn a skill and thus remove the need to delegate its achievement.

On the other side, if an agent "feels" he is overwhelmed by requests from others to exploit one of his skills, he can choose to teach it to some other agent(s) (those could be agent he has created and teached in this specific goal), to lighten his burden.

robustness If for some reason an agent has to disappear from the MAS and he owns some critical skill, he can teach it to some other agent in the MAS and thus warrant the continuity of the whole MAS.

autonomy and evolutivity During his "life" a given skill of an agent can evolve and be improved, and new skills can be added. Then the agent increases his abilities and his autonomy.

dynamic evolution When a skill of an agent need to be changed (a priori "improved"), you do not have to achieve the classical (and boring) cycle: "stop it, change source, compile and restart". The new skill can be dynamically teached to the agent (who must forget the older one). This could be particularly important for a "long life" agent who is dedicated to some role that does not withstand any interruption.

size optimisation If you consider an agent with low memory (like on organisers or cellular phones), you can chose to load your agent with only the skills required at a given time (and unload others).

2.4 Role evolution

With this dynamic evolution, the important point is that it is in fact the roles the agent can play inside the MAS that change. So the MAS in its whole evolves and can adapt according to the current flow between agents.

With this possibility of dynamic evolution of agent, you can no more use the term of "class" of agents. Even if you start from a common basis for different agents, as they can (and probably will) receive a different education due to their "experience", they will soon differ and it will eventually be impossible to consider them as belonging to a same "class". This notion has definitively no more meaning in this context. This constitutes a strong difference between such an agent oriented programming and the object-oriented programming.

2.5 Conclusion

We have presented our vision of agents built from an atomic agent through a dynamic skill learning. We will now present how this principle can effectively be applied to describe a MAS model. This will be illustrated using our own model, called MAGIQUE.

3 MAGIQUE

We will describe briefly our proposition of organisation for multi-agent systems. It is called MAGIQUE and is

[2] now if we consider the programmer as an agent, the first case is included in the second one

based on the notion of hierarchies of agents (this could be seen as well as an agent recursively made of agents, see figure 1 Mezrura et al. (1999)). More details could be found in Routier et al. (2000); Routier and Mathieu (2001)[3].

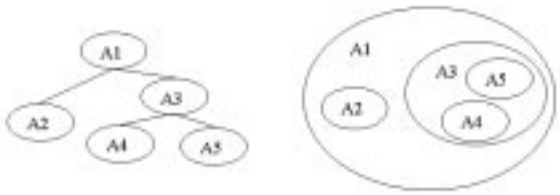

Figure 1: Two visions of the same MAS (or agent). Left: hierarchic vision. Right: "agent made of agents" vision.

The hierarchical organisation of agents allows a default automatic skill delegation mechanism that facilitates development of agents and of MAS. The used agents are the one described in section sec:atomic.

3.1 Hierarchies

A hierarchy is a tree whose root is labelled by an agent and whose childs, when any, are hierarchies too:

$$\text{MAS} = \text{Hierarchy}^*$$

$$\text{Hierarchy} = \begin{cases} \text{Agent} \\ \text{Agent} \\ \text{Hierarchy} \quad \cdots \quad \text{Hierarchy} \end{cases}$$

Leaf agents are called "*specialists*" and others "*supervisors*", these must be able (i.e. have the skill) to manage the "*team*" of agents (the sub-hierarchy) they are the root of).

A hierarchy characterises the basic structure for the routing of messages between agents. A hierarchical link denotes a communication channel between the implied agents, and when two agents of a same structure are exchanging a message, by default this passes along the tree structure. This corresponds to some *vertical* communication.

With only those communications, the model would probably be too strict. That's why, Magique offers the possibility to create direct links (i.e. outside the hierarchy structure) between agents. We call them "*acquaintance links*".

The decision to create such links depends on the agent policy. However the intended goal is the following: after some times, if some request for a skill occurs frequently between two agents, the decision to dynamically create an acquaintance link for that skill can be taken. The interest is of course to promote the "natural" interactions between agents at the expense of the hierarchical ones. These links constitute the *horizontal* communications.

Then, in Magique, there is a default communication organisation that is the hierarchy. But this structure is doomed to evolve according to the dynamicity of the MAS in order to promote the most often used relations. Then after some times, the MAS should look more like a graph.

3.2 Mechanism for skill delegation

When an agent has a task to achieve, this requires the exploitation of some skills that the agent can directly know or not. In both cases the way he invokes the skills is the same. If the realisation of a skill must me delegate to another, this is done transparently for him. This delegation is performed thanks to the hierarchical structure. Here follows the principle of skill invocation:

- if the agent knows the skill, he uses it directly,
- if he does not, several cases can happen :
  - he has a particular acquaintance for this skill, he asks him to achieve the skill for him,
  - else, he is a supervisor and someone in his hierarchy knows the skill, then he forwards (recursively through the hierarchy) the realisation to the skilled agent,
  - else, he asks to its supervisor to find for him some gifted agent and this supervisor applied the same mechanism to do it.

One first advantage of this mechanism of skill achievement delegation is to increase the reliability of the MAS: 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 MAS or is overloaded, or ...) (cf. Figure 2).

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 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 MAS, the important point is not the agents themselves but their skills.

3.3 MAGIQUE and skills

The agent used in MAGIQUE corresponds to the one presented in the section 2. They are built from an atomic agent by dynamic skill learning. Then the basic MAGIQUE agent must have the few additional skills that are

[3] and at `http://www.lifl.fr/MAGIQUE`

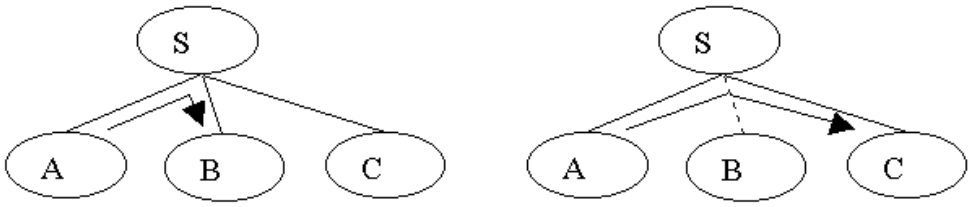

Figure 2: Agent A needs some skill , he invokes it and the request is forwarded via the supervisor S: **a.** S sends it to B – **b.** the link with B has been removed, delegation is automatically forwarded to C without A being aware of it.

particular to MAGIQUE. Skills for the hierarchy management, others for the dynamicity (connecting, creating and killing agents), are the main ones (see figure 3).

All the other skills are applications skills, it is the MAS designer task to create them and to teach them to the agents. But the MAS can be built in such a way that the learning, and the agent evolution and specialisation, are automatic. The MAGIQUE API offers the tools to do that.

3.4 Dynamicity in MAS

In MAGIQUE an important point is that MAS can evolve dynamically. This dynamicity acts at different levels.

individual an agent can acquire or forget skills. The advantages of this aspect have been discussed previously. In Magique, this corresponds to an effective exchange of skills between agents that can be remote, and the agents have the initiative of the exchange.

communication acquaintance links can be created when favourite relations between two agents occur. This creation allows to cut some repeated communications along the tree structure as communication become direct between the implied agents. The decision of creating such a link is a prerogative of the agents.

With the skill delegation principle, this contributes to a non deterministic behaviour of the MAS: two successive "execution" of the same multi-agent applications can lead to two different communication structure and consequently the skills will not necessarily be achieved by the same agents.

organisational agents can be created or removed dynamically to adapt the MAS to some constraints. Two examples among many:

- an agent is overloaded by requests about one of his skills, he can decide to create a team of agents with that particular skill and then he can delegate some requests to these agents.
- an agent can leave temporarily the MAS and recover his place later, messages for him must be stored. This can be essential for agent located on mobile computer (or phone or ...).

4 API

MAGIQUE has been put into concrete form through an API[4]. It consists in an application framework above JAVA to develop agents and multi-agents systems over heterogeneous distributed network. Agents are indeed easily built from skills through teaching of these skills from an atomic agent.

In this API, written in JAVA, primitives are provided that allow an **effective** exchange of skills between running agents. The teaching of a skill between two agents can be done with no a priori condition concerning the code: when an agent teaches a skill to a *remote* other agent, the bytecode corresponding to the skill is **really** passed from the teacher to the learner. So even, if the "class" was not initially known by the platform of the learner, everything will work fine.

Let us note, that since the API is a framework, the application part is left to the user. Thus the user must chose and create the policies used to decide when a skill must be acquire or when a new acquaintance must be created, or etc.

4.1 Agent creation

Since it exploits the skill learning, the API offers an easy framework to develop agents. Indeed, building an agent is reduced to something like a script consisting in skill "plugging" starting from atomic agent. Then in Magique the source code for creating an agent looks like:

```
import fr.lifl.magique.*;
...
 Platform p = new Platform();
 // hollow agent creation
 Agent myAgent = p.createAgent("myName");
 // agent acquires skills (= component)
 myAgent.addSkill(new SkillOne());
```

[4]It can be downloaded at `http://www.lifl.fr/MAGIQUE`

```
 myAgent.addSkill(new SkillTwo(...));
 // join a hierarchy (= MAS)
 myAgent.connectToBoss("bossName...");
...
```

As you can note in the source code, Magique uses the notion of platform and each agent must belong to such a platform. This is used to facilitate message routing between agents, but the platform offers the support for "physically" exchanging skills too. In Magique, two agents can *effectively* exchange skills with no need to make hypothesis about the location of classes: dynamic skill code loading and exchange is automatically performed (even remotely and, in this case, even if network is broken).

As an example, the agents used in MAGIQUE are built from the atomic agent through the skill acquisition/education described in figure 3.

4.2 Skill creation

In the API, building a skill is writing a class whose instances are the software components teached to the agents. The `public` methods of this component can then directly be used by the agent.

```
import fr.lifl.magique.*;
import fr.lifl.magique.skill.*;
...
public class ASkill implements Skill {
 public ASkill() {...}

 //agent will be able to use <ability>
 public void ability(...) {
 }
}
```

4.3 Skill invokation and delegation

The invokation of a skill is very simple to program too. Let us recall that in Magique, it is not necessary to explicitly know an able agent, the MAS insures that if one exists, the realisation of the skill will be automatically forwarded.

Where in OOP you should write:

```
object.ability(arg...);
```

to make a call to a given method `ability`, you must write:

```
perform("ability",arg...);
```

This has the effect that a skill named "`ability`" will be "called" (no need to know by who[5]).

The *perform* primitive is dedicated to the invokation of skills with no required answer. There exist mainly three other primitives: `ask` when an asynchronous answer is required, `askNow` for an immediate answer and `concurrentAsk` for a concurrent invokation of a skill.

[5]Of course, Magique offers also possible to precise a recipient to an invokation request if needed.

4.4 Dynamic skill acquisition

The basic MAGIQUE agent knows the skill required to acquire dynamically new skills.

Thus, if you want an agent to acquire dynamically a new skill, whose name is `"skill"`, it suffices to use `addSkill` as stated before :

```
addSkill("skill",args);
```

Now, if you want this same skill being teached by an agent names `"teacher@..."`, you need the `learnSkill` skill :

```
perform("learnSkill",
        new Object[]{"skill","teacher@...",
                     args});
```

Once this done, the agent will be able to use the new skill. This can be done even if the teacher and the learner agent are on two different remote machines. No hypothesis need to be made about bytecode, , location. If needed, the bytecode for the skill will be forwarded to the platform of the learner.

4.5 Graphical Environment

In order to facilitate the creation of hierarchies, to distribute agents over the net and to allow addition of skills to agents, we have built a graphical development tool (see figure 4). Once the skill classes have been written, this tool allows to automatically generate agents, to build a MAS with them and to distribute the MAS over a network.

Once the agents have been distributed, this environment provides an interface to track the behaviour of each agent.

A shell tool allows to interact with the agents during their "life", in order to learn them a new skill for example.

More details and some examples are available on the web site :

`http://www.lifl.fr/SMAC`

4.6 Conclusion

As you can see from this quick overview, once you know the JAVA language, there is no difficulty to apprehend and use the Magique API: the method calls are replaced by skill invocations according to a slight syntactical change explained above.

Of course the methodological difficulties due to the multi-agent aspect and the skill modularity are another problem...

5 An application to dynamic skill learning

An application of the skill exchange can be a dynamic evolution of the role of the agents: you can make evolve

```
public class Agent extends AtomicAgent {
...
/** method invoked by the constructor */
  protected void initBasicSkills()
                  throws SkillAlreadyAcquiredException {
    addSkill(new fr.lifl.magique.skill.magique.BossTeamSkill(this));
    addSkill(new fr.lifl.magique.skill.magique.system.ConnectionSkill(this));
    addSkill(new fr.lifl.magique.skill.magique.ConnectionToBossSkill(this));
    addSkill(new fr.lifl.magique.skill.magique.KillSkill(this));
    addSkill(new fr.lifl.magique.skill.system.DisplaySkill());
    addSkill(new fr.lifl.magique.skill.system.AddSkillSkill(this));
    addSkill(new fr.lifl.magique.skill.system.LearnSkill(this));
  }
 ...
}
```

Figure 3: Education of MAGIQUE agent from an tomic agent (directly extract from MAGIQUE API source code)

or reduce the abilities of an agent and then change its role in the application.

We have developed a small groupware multi-agent application consisting in a distributed conference (see Routier and Mathieu (2001)[6]). In this application, mainly two roles can be identified for the agents: the speaker and the listeners. The speaker is identified since he is the only one who own a remote control at a time. Of course, he can give this remote control to another agent and then he becomes a plain listener, and the receiver becomes the speaker. Therefore the roles of the agents can evolve dynamically.

The remote control is of course a skill since it gives special ability to its owner. Then when a speaker gives the control to a listener, what he does in fact is to teach the "control skill" to the listener and also forget it (see 5). And this is **effectively** done, the former speaker agent has really lost the ability to use the control. And this is done at the low bytecode level too: even between two remote agents, the skill bytecode is really exchanged and learned or lost.

In this case, we see that dynamic skill learning is used to manage something like the rights over an application. Roles evolve dynamically according to the skills learned or forgotten by the agents. The MAGIQUE API provides an easy to use framework to design applications with such behaviour.

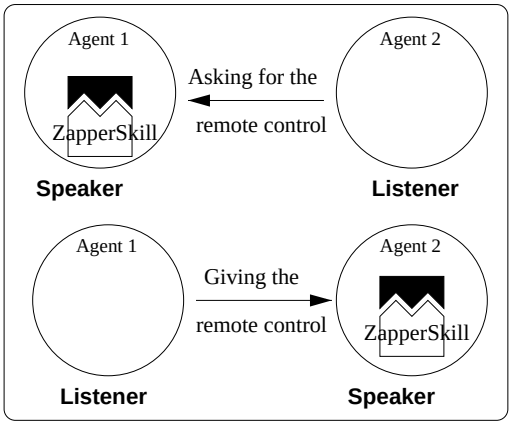

Figure 5: Dynamic evolution of roles through skill exchange

6 Conclusion

Here, we have proposed a vision of agents built from an atomic frame using dynamic skill acquisition. Skills are coherent sets of abilities and can be seen as software components. From a programmer point of view, an advantage is that modularity and reusability are promoted. Then an agent evolves during his "life". That means that he can play different roles depending on the skills he knows at that time.

This is an efficient tool to develop agent applications. The next step is to provide a methodology strongly based on the skill notion whose formalisation must be deepened. Starting from studies like Wooldridge et al. (1999), a fundamental consideration of the nature of interaction must be undertaken. Interactions can indeed be tackled in terms of skills and roles. We are working on a formalism for representing them and we think it will lead to an automatic generation of MAS and agents. Dynamic learning should help that.

An API has been developed to validate these ideas. It allows to effectively build agents by dynamic "skill plugging" and to distribute them, no hypothesis about where the source of a skill is need to be done as soon as the learner knows it. The API, the graphical environment, some small illustrating examples and a brief tutorial can be downloaded at:

`http://www.lifl.fr/SMAC`

[6] or/and have a look at `http://www.lifl.fr/MAGIQUE/examples/diapoExt.html`

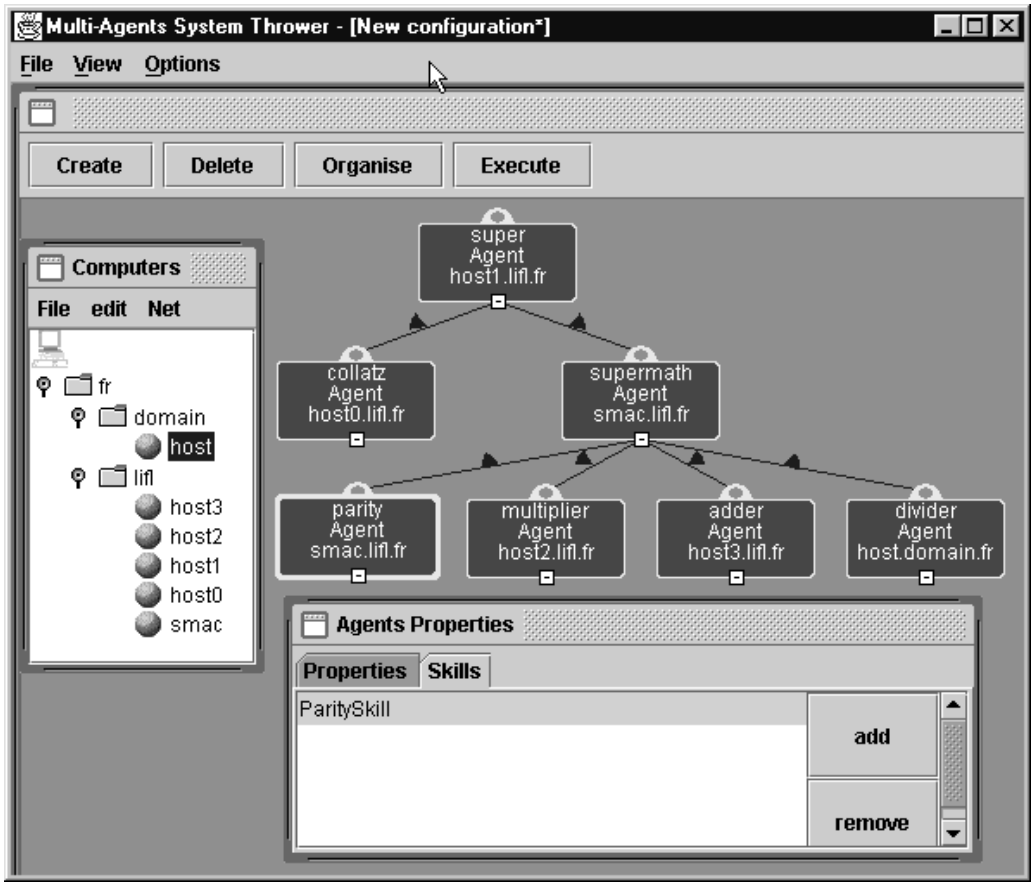

Figure 4: Graphical environment for building and distributing agents.

References

N.E. Bensaid and P. Mathieu. Un modèle d'architecture multi-agents entièrement écrit en prolog. In *IV Journées Francophones de Programmation Logique, JFPL'95*, pages 381–385, Dijon-France, 1995. teknea, Toulouse-France.

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

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.

S. Franklin and A. Grasser. Is it an agent, or just a program?: A taxonomy for autonomous agent. In *Proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag, 1996.

B. Horling and V. Lesser. A reusable component architecture for agent construction. Technical report, UMass Computer Science, 1998.

C. Mezrura, M. Occello, Y. Demazeau, and C. Baeijs. Récursivité dans les systèmes multi-agents : vers un modèle opérationnel. In *JFIADSMA'99*, pages 41–52. Hermès, 1999.

JC. Routier and P. Mathieu. Une contribution du multi-agent aux applications de travail coopératif. *TSI Hermès Science Publication. Réseaux et Systèmes Répartis. Calculateurs Parallèles*, 13. Numéro Spécial : Télé-applications, 2001.

JC. Routier, P. Mathieu, and Y. Secq. Dynamic skills learning. Technical Report 2000-06, LIFL, 2000.

Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

M. Travers. *Programming with Agents: New metaphors for thinking about computation*. PhD thesis, MIT, 1996.

M. Wooldridge, N. R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. 1999.