Towards MAS's design regarding communications and mobility

Gilles Klein, Nejla Amara-Hachmi, Amal El Fallah-Seghrouchni

LIPN-UMR 7030
Université Paris 13
99, Av. J.B. Clément
93430 Villetaneuse, France
Phone: (+33) 1 40 40 35 78
{gk, na, elfallah}@lipn.univ-paris13.fr

Abstract. In this work, is shown how software engineering and mobile agents may help the building of distributed-problem-solving programs. This paper provides guidelines to build multi-agent systems. It focuses on a method to distribute agents among computers on network-based systems in order to improve load-balancing and reduce communications' cost. Mobile agents are used to improve the runtime load distribution.

Keywords. Design methodology, load balancing, distributed systems, Petri nets, mobile agents.

Conference Topics. Multi-Agent Systems and Distributed AI

1 Introduction

The most important characteristic of agents is their lack of memory sharing. It makes them relatively easy to dispatch over a network. However, even if agent-oriented design simplifies this distribution, it is still difficult to realize. The first difficulty a software designer may encounter (following a chronological order) is the recycling of former code-components. Integrating “non-agent” code into a Multi-Agents System (MAS) is not a simple task [JEN 93]. To overcome this problem, the designer must differentiate the tasks “to agentize” in a recursive way. This gives some choice over the granularity of the decomposition so that we can consider the software components to be recycled (components cannot normally be cut in different parts).

The second difficulty is the management of the risks associated with synchronization. This difficulty must be solved by a formal analysis of the MAS [Attaiya 1998].

To fill these two necessities (formalism and recursiveness), we used recursive Petri nets to decompose the program in sub-tasks.

The third difficulty, commonly associated with distributed systems, is to manage the load-balancing. It can be shown that a static load-balancing (when the tasks have been distributed prior to the runtime) does not always give maximum efficiency , even if it was theoretically optimal. This is due to the approximations done when modeling a computer system [Taylor 2001]. So dynamic load-balancing is needed to correct the imperfections during the runtime. To perform dynamic load-balancing, we conceived a system based on MASIF standard. As written above, agents, because of their independence, are relatively easy to distribute over a network, so they are very-well adapted to help solve load-balancing problems. However, their independence may render the situation more complex as no agent has a global knowledge of the system. Now, the goal of load-balancing is not the improvement of the performances of one agent but of the whole MAS. So it has been decided to use a method of preferences aggregation based on negotiation [Vauvert 2001] to redistribute tasks during the runtime.

2 Overview of the method

Our method of conception and implementation of MAS is composed of five different steps (see Fig.1). During the analysis step, the designers must find the specific pieces of information which will be necessary to create the system and then analyse them. This step can be decomposed into two different parts:

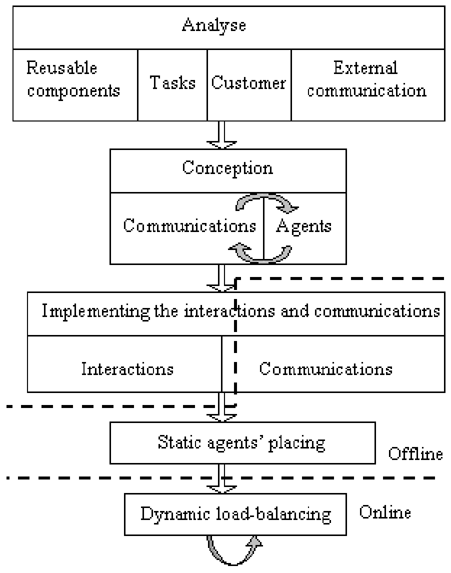


Fig. 0. Method of conception and implementation of MAS.

- The collect of information about the recyclable code, about the needs and tools of the customer, as well as the analysis of the future use of the program and so of its needs of communication with the outside (users, other programs…)
- The analysis of the task; during which, the task of the program must be “cut” in different subtasks that will then be given to different agents.

Afterwards, during the design phase, the agents, themselves, are designed.

Once the agents have been designed, their interaction modes and then their communication systems (down to the lowest level of abstraction). During this step, the implementation of the system, itself, begins.

The last two steps include the positioning of the agents on the computer system (as we consider *a priori* that the MAS will run on a multi-processor system (for example a PC-cluster). The agents must be placed but they will also have to be movable in order to correct errors made during the static step or in order to adapt to an evolution of the system.

3 Multiagent DOGS

The method presented in this article is born of our collaboration with THALES on the “Multiagent DOGS” project. So it is necessary to describe DOGS so that the decomposition step of the method can be explained.

DOGS is a system of electronic circuits diagnostic based on intervals-logic [Taillibert 1997]. The former versions of DOGS were not built using the agent-paradigm. So it has been necessary to decompose the original program into different sub-tasks which could then be distributed between the agents. We had to reuse the code of the former versions as much as possible. To obtain this result, we had to define the minimal granularity so that we would not break reusable software-components. There were also tasks which were sequential so separating them would not bring any gain.

DOGS can be separated in several distinct parts: the program starts with a collect of information (measure of the voltage value near certain electronic components (switches, diodes for example). Then the characteristic equations of the circuit are calculated at different levels of abstraction (from high level functional blocks to basic component level). Then the theoretical values associated with the circuit are calculated and compared to experimental ones.

4 Design and implementation of the system

4.1 Decomposing of the task between the agents

4.1.1 Analyzing the recyclable software-components

Before building the MAS, the designer should begin by analyzing the potentially recyclable software components. We do not advice one method or another for this step of the conception. The most important during this step is to ensure that these components are well-defined and constructed. It is necessary as, during the next step, the granularity of the decomposition is partly determined by the results of this analysis.

4.1.2 Building the Petri net

We will, now, describe the decomposition of the program into agents. We will use recursive Petri nets to achieve this goal. During the first part of the decomposition

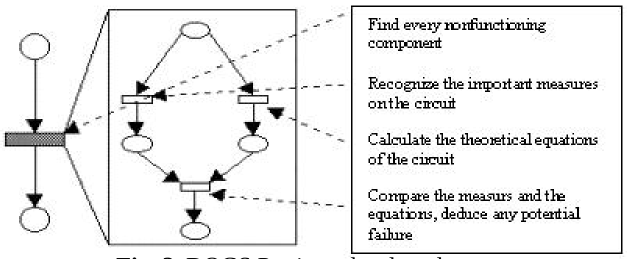

Find every nonfunctioning component
Recognize the important measures on the circuit
Calculate the theoretical equations of the circuit
Compare the measurs and the equations, deduce any potential failure

Fig. 2. DOGS Petri net developed once

phases, the designers must build the application's Petri net. A top-down method, based on recursive Petri nets, is used (Figure 2). The Petri net representation of the program which is obtained is not unique and it can be necessary to backtrack in order to make the recyclable components match the transitions of the Petri net.

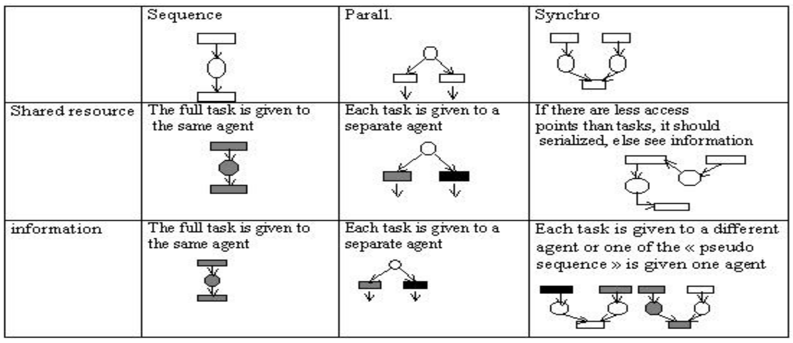
The figure 2 could present the first iteration of DOGS's Petri net design.

4.2 Building the agents and the MAS

4.2.1 Distributing the tasks between the agents through recursive Petri net

Once the application's Petri net has been built, it is used to build the agents. It differentiates the tasks to be realized (they are represented by the transitions of the Petri net) and it also gives the links between them. Two kinds of relations between two tasks must be separated : there are "indifferent" and synchronized tasks. (indifferent tasks are not related). Two indifferent tasks can be given to two different agents so they can be fulfilled concurrently. When two tasks are synchronized, they must be distributed following a rule, as, for example, if they are automatically sequential they should be given to the same agent. We propose the rule given by the figure 3. It is advised to distribute the parallel and indifferent tasks on as many agents as possible as it will then simplify the load-balancing on a multi-processors system.

The are also two other criteria that must be considered, the software components and the final clarity of the code. The first one's concern is the recyclable code (or the functions libraries). It may happen that two tasks without any link have to be given to the same agent as the code fulfilling both of them is inseparable (for example, if they use the same sub-functions or if they are part of the same commercial library).



	Sequence	Parall.	Synchro
Shared resource	The full task is given to the same agent	Each task is given to a separate agent	If there are less access points than tasks, it should serialized, else see information
information	The full task is given to the same agent	Each task is given to a separate agent	Each task is given to a different agent or one of the « pseudo sequence » is given one agent

Fig. 3. Task distribution between agents regarding the tasks' relations

In DOGS' case, we decided to separate the study and the research of the functional equations of the circuit between “Block Agents”. Those agents could have been distributed randomly on the circuit, but instead, we decided to create an abstraction-based hierarchy of Block Agents for clarity and programming-difficulties reasons. So there are high-abstraction Block Agents that study the circuit on a function-block level (considering logical doors, for example) which then distribute more mundane agents on each suspect function block. The second category of agents studying the circuits components (transistors, diodes ...), can find the non-functional components or verify that everything works correctly. It was also decided to centralize every calculus tasks into a single category of agents (instead of placing it in the Block Agents) because of the calculus libraries.

4.2.3 Agents' building

In the last step, we distributed the tasks between the agents, however, it is not sufficient to completely describe the agents. We shall now describe the method of construction of the models of agent. Abstractly, we could consider that an agent is the union of three parts: a work-module, a memory and a communication-module. In what follows, no rule of the memory and work-module is describe, it could not encompass every specific situation. Our concern here is how to describe the MAS using Petri-nets so as to not lose the advantages of this formalism, once the system is built. In what follows, the work-module (WM), which includes every tasks given to the agent in the precedent part, will be represented by a set of Petri nets and the memory (MEM) by the local environment of these networks.

Agent's WM's Petri nets are different from the nets of the tasks that have been assigned to them. The difference is caused by the independence of the agents, the

WM's environment is the MEM of its agent when the original tasks were encompassed within the program's global environment.

So the new representation must take communications into account as the tokens and the environment data must be transmitted from one agent to another. The Petri nets must be modified to manage the communications and make them verifiable. To obtain that result, we represent communications as in the figure 4.

In the example described in the figure 4, the original Petri net was *a choice*, it was separated into two different agents. Two types of communications had to be defined, one for the data (type **a** on the figure) and one for the "token communications" (type **b** on the figure). We also had to add new transitions to represent the agent's management of the communications.

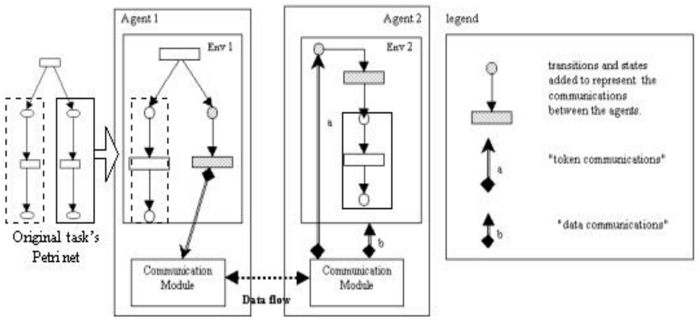


Fig. 4. a Petri net-based model of communications

The model we obtained takes communications into account.

4.3 Building the communications and adapting the agents

Once the agents have been built, their communication module must be conceived. Two points must not be overlooked:

First, the designers must consider the level of abstraction of the communications.

Then, they must choose the protocols of communications to be used.

FIPA ACL-based messages slow the system when compared to JAVA object transmitted between Java-programmed agents (as the first kind must be transformed into typed data before being used. To choose the communications modes, having studied well the needs and the links between the agents and the outside is necessary [KLEIN 2002].

There are also different communication vectors for the messages (e-mails using TCP-IP to UDP datagram sockets which are considered for DOGS). Choosing a mode of communication depends on a lot of criteria (security, compatibility…). To choose TCP "because anyone does so", as in APRIL [Mc Cabe 1994], is not a good solution.

Once the communications are built, the agents must be distributed on the computer system, and the weight of the communications must be calculated (at least very approximately, for example by using the following formula)

$E*(T1+Tcom+T2)$ (E is the estimation of the frequency of this specific kind of communication, T1 the time necessary from sender agent to build the message, Tcom the average length of the transport from Emission to Reception and T2 the length of the treatment necessary for the receiver to use the content of the message).

If a communication is too costly, it must be diminished by placing the communicating agents near to one another or eliminated by breaking the communicating agents and placing the two communicating tasks in the same agent.

4.4 Placing the agents

Placing agents on a network so as to optimize the computational power and minimize the weight of the communications is difficult. To solve it, simplifying hypothesis must be done. The most usual is the nullity of the cost of communications which is unacceptable in our situation. So it is necessary to find good hypothesis and then to apply operational research methods from TABOU to column generation [Desaulnier 1998] depending on the difficulty of the problem.

5 Dynamic load distribution : use of mobility

In the previous study, we proposed a method to build agents and distribute them among computers in order to minimize their costing communications and balance their tasks. This distribution is static as it is done *a priori* during the design step, so there might be some limitations that can arise during the runtime. In this section, we'll present a way to balance load dynamically using mobile agents.

5.1 Why mobile agents?

One of the most difficult steps of the proposed MAS designing method is the physical distribution of the agents. In fact, the designer has to make a distribution that reduces the cost of the interactions and follows two main principles:

- Agents should always have enough computational power to work correctly. Therefore they must be assigned to computers where the computational power can be distributed between them proportionally to their needs.
- Agents that need to communicate frequently together should be placed so that the cost of connections is minimal, which may mean that the best solution is to assign them to the same computer.

These principles can be contradictory, so the designer has to look for the best compromise. Unfortunately, in spite of the design effort to achieve the best solution, some problems may occur during the execution and induce a fall down of the system’s performances. Indeed, as we deal with open systems, the number of agents necessary to solve a problem may be unknown at the beginning and can even change during the execution process as well as the amount of communications between them. For example, in the DOGS system, the diagnosis rests on the model based diagnosis which follows the hierarchy of the electronic circuit representations (models). The diagnosis is hierarchical and top down, i.e. it starts from the most abstract model of the electronic circuit down to the most basic models (the components of the circuit). To check hierarchically the models, agents are created dynamically at each level of diagnosis. The distribution process should then take into account these changes during the runtime.

For these reasons, it is advantageous to seek for a dynamic distribution that fits with changes which may occur in the runtime. In fact, the agents’ distribution should be adjustable so that it can be as adapted as possible to situations like those listed above. To meet these requirements, we propose to use mobile agents because they can migrate from one computer to another to reach a distribution that minimizes the communications’ cost, they reduce the need for network availability and they increase fault tolerance [Illmann 2000]. Yet, before migrating, agents need to know when and where to go. To take this decision, the state of the whole system (e.g. the number of exchanged messages, the load on each machine, etc.) has to be considered. That's why we need a global view of the system, but we should avoid centralized solutions as they are not tolerant and don't fit the distributed aspect of our problem. We should not also let each agent take the migration decision by itself because it would have to negotiate with all the other agents which may increase more and more the number and the cost of communications. Consequently, decisions should be taken neither by agents nor by one central component but by a medium level between them which we will present in the next section.

5.2 The migration's decision

5.2.1 Need for a runtime environment

In order to move correctly, mobile agents require a specific and secure runtime environment on all potential hosts. This environment has to provide a support for agents' management, execution, localization, migration, communication and security control. Another fundamental requirement is interoperability between different agents' platforms. An agent platform compliant with the OMG MASIF standard [Crystaliz 1997] may be able to fulfill these requirements. In fact, MASIF adopts concepts of agent systems (i.e. agencies), places and regions. A place groups the functionality within an agency, encapsulating certain capabilities and restrictions for hosted agents. Each agency comprises at least one place in which the hosted agents are running. A region facilitates the platform management by grouping sets of agencies that belong to a single authority [Integrating Mobile Agent Technology and CORBA Internet]. Two interfaces represent the core of the MASIF standard: the MAFAgentSystem interface which is associated with each agency to provide operations for the management and transfer of agents, and the MAFFinder interface which associated

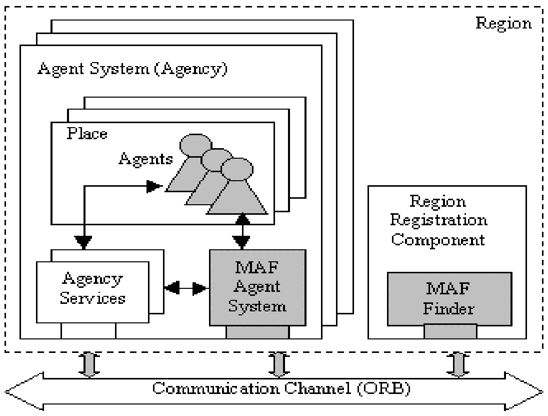


Fig. 5. MASIF Compliant Platform Architecture [Crystaliz 1997]

with every region to support the localization of agents, agencies and places (see Figure 5).

5.2.2 Migration actors

The goal of an agent's migration towards a new host is to achieve, when it is possible, a better state of the hole system. In the previous section, we argued that the migration's decision has to be made neither by each agent on his own nor by a central component, but by a medium level which can cooperate with other agents to obtain a global view of the system's state. In this work, we propose to provide each agency with specific stationary agent, a *manager*, that has to observe the work of agents hosted in the agency and decide, in concert with other *managers*, on eventual migrations (see Figure 6).

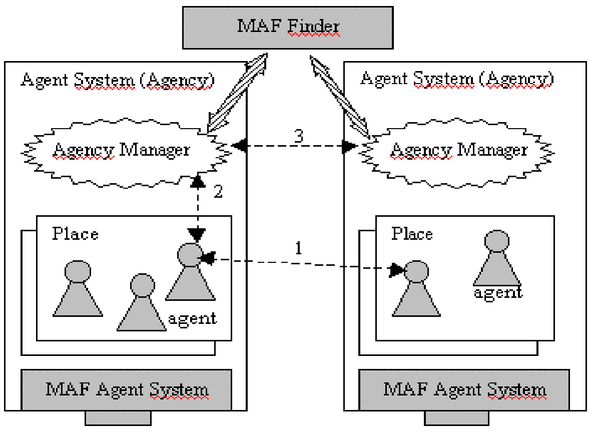


Fig. 6. Migration decision

In the static step (described in section 3.4), once agents assigned to machines, everyone would be able to recognize agents that communicate with him (its contacts). During the execution time, every agent should keep a local trace of his communications' frequency, intensity and partners [Mazouzi 2002]. So, the request for migration can be triggered by two ways :

1. As every agent has a local trace of its communications, it can distinguish the most costly and frequent ones. It contacts then the *manager* of its agency and requires to move "near" the distant agent with whom it has the most important communications. Then, this *manager* contacts the *manager* of the agency where is hosted the distant agent and tries to manage with it the best way to make the agents closer to each other. Nevertheless, the decided migration should take into account the performances of the machines, their capacity to receive new agents and shouldn't damage the other communications. The migration decision is based on a consensus (agreement) to be reached be the managers according to a method proposed in [Vauvert 2001].
2. Periodically, the agents send their communications' local trace to the *managers* of their hosting agencies. Therefore, the *managers* can have a global trace of all communications that they analyze to manage the appropriate migrations. In order to achieve a consensus between the managers, we use the method reported in [Vauvert 2001].

In both cases, *managers* should be in contact with the MAFFinder interface that enables them to locate the region's components. For example, when a *manager* receives an agent's request to get closer to another agent specified by its name, the *manager* consults the MAFFinder that returns the agent's location.

5.3 Realization

To implement this work, we started with the migration module. To migrate, the agent’s code, state and execution information have to be moved and restored at the new location. The mobile agent’s community distinguishes between two types of mobility: strong mobility that means the transfer of the agents code and its complete state, and weak migration which transfers only the agent’s code and data [Picco 2001]. Despite the advantages of strong migration (it is transparent, the execution resumes after the migration instruction, etc.), in our implementation of migration process, we opted for weak migration. This can be justified as we use JAVA programming language which offers many advantages sought by mobile agents' programmers (such as platform independence, multi-threading, network APIs, etc). In fact, the Java Virtual Machine (JVM) provides only mechanisms sufficient to implement weak migration (namely the ability to program the dynamic class loader) but insufficient to deal with the execution state [ObjectSpace, Inc, (1997), Voyager].

Our agents are supposed to be collections of java classes. To make them move, we used Class Loaders which enable the JVM to load classes without knowing anything about the underlying file semantics. The abstract class, *ClassLoader*, is a subclass of *Object* and is contained in the *java.lang* package. Our application inherits from the *ClassLoader* abstract class and extends its functionality to load classes dynamically. The JVM loads classes from the directory defined by the CLASSPATH environment variable on the local file system or from remote destinations over the network. For example, we implemented a agent *manager* which is process running all times, listening for requests from agents who would like to migrate to its agency. This process is based on threads and communicates via TCP-based steam network connections. Both the *manager* and the agent that wants to migrate use sockets to implement reliable stream connections. The manager uses a socket class to accept connections from agents. Whenever an agent connects to the port number on which the *manager* is running, a new socket object, connected to some new port, is allocated to the agent to communicate through. The *manager* goes back then listening for more requests. Once an agent opens a connection with the *manager*, it sends the URL for which the code is to be executed.

6 Conclusion

In this paper, we presented a top-down method of conception and implementation of DPS-oriented MAS. Our goal is to emphasize the advantages of the MAS on matters of load-balancing, particularly during the implementation phase.

Here, agents are fully cooperative, as they have to transmit every information concerning their communications to the managers. But, further, our goal is to be able to apply the same method to solve the problem of computer resources used by several different users. It will present several new difficulties as the agents will not be cooperative anymore and will try to optimize their own situation even at the other agents expense.

We also want to focus on the changes that will be necessary for our method to be adapted to the implementation of MAS on open computer systems open (as for example an association of PC users connected by the Internet). This last perspective raises new problems, not only for security reasons but also because it will introduce heterogeneity among the computers connected, this difficulty may be simplified by the use of JAVA virtual machine.

Reference

Attaiya, H., Welch, J.., « Distributed Computing », 1998,Mc GrawHill

Baumann, J., Hohl, F., Rothermel K. and Straßer M.), « Mole - Concepts of a Mobile Agent System », 1998, *World Wide Web*, Vol. 1, Nr. 3, pp. 123-137

Brenner, W., Zarnekow, R., Wittig, H., Intelligence Software Agents • Foundations and Applications; 1998, Springer

Crystaliz Inc, (1997), General Magic Inc, GMD FOKUS, IBM, TOG, *OMG Joint Submission: Mobile Agent System Interoperability Facility* ,available via ftp://ftp.omg.org/pub/docs/orbos/97-10-05.pdf

Desaulniers et al., « A unified framework for deterministic time constraint vehicle routing and crew scheduling problem » *fleet management and logistics*, p57 93, édité par Crainic et Laporte, 1998 Kluwer Boston

Eck van, P.. (1996), « The DESIRE Research Program », http://www.cs.vu.nl/vakgroepen/ai/project/desire/

Elfallah-Seghrouchni, A., Haddad, S.. « A Recursive Model for Distributed Planning ». *Proceedings of ICMAS'96*. 1996, AAAI Press.

FIPA, (2001), FIPA Agent Software Integration Specification, http://www.fipa.org/specs/fipa00079/XC00079B.htm

FIPA, (1998), FIPA 98 Draft Specification: Part 11: Agent Management support for Mobility, FIPA 8415, Version 0.3, Foundation for Intelligent Physical Agents.

Grasshopper: A Platform for Mobile Software Agents
http://www.grasshopper.de/download/doc/GrasshopperIntroduction.pdf

Integrating Mobile Agent tech and CORBA http://www.det.ua.pt/Project/difference/work/D7/d7chap4.html

Klein, G., El Fallah-Seghrouchni, A., Taillibert, P., « HAMAC: an agent-based programming method », 2002, (to be published in the *proceedings of AAMAS 2002*). AAAI Press.

Knuth, D. E., « The Art of Computer Programming », 1998, Addison-Wesley Pub Co

Lange, D., Oshima, M., The Aglet book « Programming and Deploying Java Mobile Agents with Aglets », Addison-Wesley, http://cseng.awl.com/bookdetail.qry?ISBN=0-201-32582-9&ptype=0

Illmann, T. Krüger, T., Kargl, F., Weber, M., « Migration of Mobile Agents in Java : Problems, Classification and solutions ». In: *Proceedings of MAMA ' 00*,2000, Wollongong, Australia.

Jennings, N. R., Varga, L. Z., Aarnts, R., Fuchs, J and Skarek, P., « Transforming Standalone Expert Systems into a community of Cooperating Agents », 1993, *Int. Journal of Engineering Applications of Artificial Intelligence* 6 (4) 317-331

Jennings, N. R., “Agent-Oriented Software Engineering”, *Proceedings of MAAMAW'99, 1999,* LNAI 1647 :

Loiez, E., Taillibert, P. Polynomial Temporal Band Sequences of analog Diagnosis, IJCAI'97 Nagoya. August 23-29 1997

MASIF : The OMG Mobile Agent System Interoperability Facility
http://www.hpl.hp.com/paersonal/Dejan.Milojicic/mace.pdf

Mc Cabe, F. G., Clark, K. L..(1994), « April - Agent Process Interaction Language ».

Mazouzi, H., El Fallah-Seghrouchni, A., Haddad, S..(2001), « Open Protocol Design for Complex Interactions in Multi-agent Systems », (to be published in the *proceedings of AAMAS 2002*).2002, AAAI Press.

ObjectSpace, Inc,, « Voyager : The agent ORB for Java. Core Technology User Guide », 1997.

Picco, G. P., « Mobile Agents: An Introduction ». In: *Journal of Microprocessors and Microsystems*, vol. 25, no. 2, 2001, pp. 65-74,. Invited contribution in a special issue on mobile agents edited by A. Corradi.

Taylor E. et al., « Balancing Load versus Decreasing Communication: Parameterizing the Tradeoff ». *Journal of Parallel and Distributed Computing*, Vol. 61, No. 5, 2001, p.567-580

Vauvert, G., El Fallah-Seghrouchni, A., « E-commerce Agents », in *the Proceedings of the 2nd Intelligent Agent Technology*, 2001, 355-356, World Scientific Publishing Co.

http://www.meitca.com/HSL/Projects/Concordia/