From Analysis to Deployment: a Multi-Agent Platform Survey

Pierre-Michel Ricordel and Yves Demazeau[1]

LEIBNIZ laboratory, 46 av. Felix Viallet, 38000 Grenoble, France
{Pierre-Michel.Ricordel, Yves.Demazeau} @imag.fr

Abstract. This paper presents a survey on multi-agent platforms, with a particular focus on methodology. It presents the four stages of the construction of a multi-agent system and derives from these stages several criteria for comparison. These criteria are then used to evaluate four selected platforms, and lead to a discussion on the future of multi-agent platforms.

1 Introduction

Recently, natural evolution of Multi-Agent Systems (MAS) has leaded them to migrate from the research laboratories to the software industry. This migration is good news for the entire MAS community, but it also leads to new expectations and new questions about multi-agent system methodologies, tools, platforms, reuse, specification, and so on. This introduction of Software Engineering techniques into Multi-Agent Systems gains more and more interest in both the research area and the industry. The best example is the brunch of announcements about new multi-agent platforms, usually supposed to be the ultimate tool to build multi-agent systems. Since this kind of affirmation seems far-fetched compared to the difficulty of the problem, we need some elements to evaluate and to compare multi-agent platforms. In this paper, we propose an analysis grid, built from criteria that evaluate each of the stages encountered during the creation of multi-agent systems. Of course, as in benchmarks, any criteria is relevant to a specific outside need, and a platform can only be compared relatively to another one because there is nothing like an absolute measurement scale for multi-agent platforms. But some advance can be made in platform comparison by proposing comparison criteria and applying them to multi-agent platforms as in this paper. In section 2 we present the MAS construction stages, then, in section 3, the evaluation criteria drawn from these stages. In section 4 we evaluate four multi-agent platforms (AgentBuilder, Jack, MadKit, Zeus) with the established criteria, and finally we discuss the results in section 5. Since the multi-agent platform market is evolving very fast, we would like to insist on the fact that the platforms analysed in this paper are evaluated given their current state at the time of writing this paper (first semester 2000), and that their specifications may change as time goes on.

[1] Yves Demazeau is research fellow at CNRS (Centre National de la Recherche Scientifique).

2 Four Stages to Build Complex Software

Multi-Agent Systems are complex software. Building such complex systems software requires using adequate engineering methods. Traditionally, software engineering distinguishes three stages of construction: Analysis, Design and Development. We believe that for systems like Multi-Agent Systems this is not sufficient, and that describing the entire MAS creation process, from early design to a running application, we have to distinguish a fourth stage after the development stage, called deployment.

We also know that the exact frontiers between the three classic stages (analysis, design and development) are still subject to debate in software engineering. Since we believe that these stages are just quantified levels along a natural conceptual continuum, we propose to use the following definitions for these four stages:

- Analysis: the process of discovering, separating and describing the type of problem and the surrounding domain. Practically, it consists of identifying the application domain and the key problem.
- Design: the process of defining the solution architecture of the problem, in a declarative way. Practically it consists of specifying a solution principle of the problem, for example using UML [1].
- Development: the process of constructing a functional solution to the problem. Practically it consists of coding the solution with a particular programming language.
- Deployment: the process of effecting the solution to the real problem in the given domain. Practically it consists of launching the software on a network of computers, and then, to maintain or to extend its functionality.

The two former stages are more involved with methodology and the two latter addresses more technical aspects. In this paper we focus on the different methodological stages, but the software development process support is also an important issue.

3 Criteria of Examination

In this part we will present the different criteria we have drawn from the development stages presented before.

3.1 Four Qualities of Development Stages

The process of drawing common *qualities* from the four building stages is not evident. By *quality*, we mean a distinct feature that characterises a positive or a negative aspect in the practical realisation of a particular stage. We have determined four qualities that seem relevant to us to all the stages, and which address as many aspects as possible of practical development pitfalls. These are:

– Completeness: The degree of coverage the platform provides for this stage. This addresses both the quantity and the quality of the documentation and tools provided.
– Applicability: The scope of the proposed stage, in other words the range of possibilities offered and the restrictions imposed by the proposed stage.
– Complexity: The difficulty to complete the stage. This includes both the competence required from the developer and the quantity of work the task requires.
– Reusability: The quantity of work gained by reusing previous works.

In the next section, we will apply these qualities to the four stages of development.

3.2 Application of the Qualities to the Stages

The four qualities applied to the four stages of development result into sixteen criteria for evaluating any platform.

Analysis The Analysis criteria includes:
– Completeness: is the analysis method useful, and is it well documented?
– Applicability: to which domains and problems does this method apply?
– Complexity: is the analysis method easy to understand and easy to apply?
– Reusability: is there a way to exploit previous analysis of similar problems or of similar domains? Are there analysis examples supplied?

Design The Design criteria includes:
– Completeness: is the design method useful and well documented? Are there tools to support the design process?
– Applicability: what kind of MAS can be designed by this method?
– Complexity: is the design method easy to understand and to apply?
– Reusability: is there a way to reuse existing designs? Are there useful designs supplied?

Development The Development criteria includes:
– Completeness: How useful are the supplied development tools?
– Applicability: Is there some functionality impossible to achieve with the development tools?
– Complexity: Are the development tools and languages easy to use? How popular is the language used?
– Reusability: Is there an active support to reuse the code?

Deployment The Deployment criteria includes:
– Completeness: Is there a support for the deployment of the MAS?
– Applicability: Does the deployment tool support visualisation, profiling, on-line maintenance, etc?
– Complexity: Are the deployment tools easy to use and understand?

– Reusability: Is it possible to integrate a previously existing agent dynamically into a new multi-agent system without any modification?

3.3 Other Criteria

We can also consider some other practical criteria that one has to take into account when choosing a platform. In particular, these criteria can condition the adoption of a platform to a particular project:

– Availability: Is there a trial version, confidential clauses, is the source code available? How much does it cost?
– Support: What are the future developments of this platform? Is this platform used in large scale?

At a finer level, we could explore more technical aspects, such as process handling, mobility, security, standardisation, but we limit our analysis to a more conceptual level.

4 Platform Examination

4.1 Choice of the Platforms

We have chosen a set of platforms that have in common:

– to be popular and regularly maintained (for bug fixes and extra features),
– to be grounded on well-known academic models,
– to be developed with industry-like quality standards,
– to cover as many aspects as possible of Multi-Agent Systems, including agent models, interaction, coordination, organisation, etc.,
– to be simple to set-up and to evaluate. This includes good documentation, download availability, simple installation procedure, and multi-platform support.

We have deliberately avoided platforms that:

– are still in experimental state, abandoned, or confidentially distributed,
– are grounded on vague or non-existent models,
– cover only one aspect of multi-agent systems, like single agent platforms, mobile agent platforms, interaction infrastructures toolkits,
– Are too short on some construction stages, like purely methodological models or development tools without methodology.

We would like to lay stress on the fact that in this study we have deliberately chosen platforms coming from the Multi-Agent community towards Software Engineering, and not platform coming from the Object-Oriented, Component-Oriented or Software Engineering community towards Agent systems. This includes popular mobile agent platforms, such as Voyager, Grasshopper and others. This kind of platform has very

good deployment tools, but generally does not address at all Multi-Agent specificities, especially at the analysis and development stages.
Given these criteria, we have selected four platforms to be presented in this paper:
– AgentBuilder®
– Jack™
– MadKit
– Zeus

4.2 Commented Evaluation

In this section, we evaluate the four selected platforms, using the above mentioned criteria, each platform is followed by a short comment.

AgentBuilder® (`http://www.agentbuilder.com/`)
AgentBuilder is an integrated tool suite for constructing intelligent software agents. It is developed by Reticular Systems Inc., and is grounded on the Agent0 [2][3] and Placa [4] BDI models. This tool is remarkable both by the high quality of its software and the well-known academic background model used. The global methodology is shortly described in the AgentBuilder User's Guide [5].

Analysis The analysis stage consists of the specification (in OMT) of the objects in the domain and the operations they can perform, followed by the production of a domain ontology. Graphical software tools support this stage, but very few hints are given about how to perform a good analysis. This kind of domain decomposition is almost applicable to any domain/problem. To complete this stage, a good experience in object modelling is sufficient, and the provided tools are quite intuitive to use. On the reusability aspect, AgentBuilder supports the reuse of ontologies through an ontology repository.

Design The design stage consists of the decomposition of the problem into functions that agents may perform. Then the agents are identified, their roles and characteristics are defined and finally the interaction protocols that they use are designed. Here also, efficient software tools (the Agency Manager and the Protocol Editor) are provided, but few indications are available to achieve a good design. This stage is applicable to the design of any multi-agent system composed of intelligent agents. In order to complete this stage, a solid background in multi-agent design is preferable, but here also, the tools are easy to use. The reuse of protocols is possible with the protocol repository.

Development The development stage consists of defining the behaviour of agents, more precisely the behavioural rules, initial beliefs, commitments, intentions and capabilities of agents. It is also during this stage that external actions and agent Graphical User Interfaces are integrated into agents, using the Project Accessory Class library. Graphical software tools support all these tasks. This stage is only applicable to agents that use the proposed BDI architecture. A background in logic programming and BDI models is needed, but the proposed BDI model is a classical one, and the graphical tools simplify the construction of behavioural rules, avoiding syntactic and semantic errors during their construction. Finally, agents and external actions can be reused across projects.

Deployment A Run-Time Agent Engine, that interprets agent programs, executes every agent generated during the previous stages. A debugging tool is also provided, allowing to monitor the agents' mental model step-by-step, and to monitor agent's interactions. Due to the full BDI agent model, only MAS composed of a few cognitive agents are achievable. The complexity of deployment is reduced by graphical and intuitive tools.
Agents may dynamically be added to a running multi-agent system, but this feature is not directly supported by the tools.

Other AgentBuilder is a closed-source commercial product. A free evaluation version is available, limited to the tutorial agents. It is available in three versions: AgentBuilder Lite, AgentBuilder Pro and AgentBuilder Enterprise. Academic versions are also available. At the time of writing this paper, license fees range from $100 to $5000 US$ depending on the version. Phone and electronic support is available. The product is still under evolution.

Comments AgentBuilder's documentation covers almost all the stages, from analysis to development. This is a good point, even if the analysis and design parts are quite succinct. They deal more with *what to do* rather than with *how to do*, which is more difficult to determine but is more helpful to the developer. Another good point is the software tool, which cover almost all the aspects of the stages, and establish links between them (as the automatic generation of behavioural rules from protocol definitions). The disadvantage of such a frame is that it limits the versatility of the tool. Multi-Agent Systems constructed with AgentBuilder are homogeneously composed of agents that use the AgentBuilder agent model. There is no easy way with AgentBuilder to integrate agents that use another model, or to interact with an environment. As any complex tool, AgentBuilder is long and difficult to learn, but once the tool mastered, can become very productive.

Jack™ (`http://www.agent-software.com.au/`)

Jack is described as *an environment for building, running and integrating commercial JAVA-based multi-agent systems using a component-based approach.* It is developed by Agent Oriented Software Pty. Ltd., an Australian commercial company. It is based on the BDI model dMARS developed at the Australian Artificial Intelligence Institute (AAII) [6]. Jack focuses mainly on the development stage. Analysis and design steps are only mentioned in [7]. The toolkit consists of the JDE (Jack Development Environment), a graphical tool to manage projects, the Jack Agent

Language (JAL) compiler, that translates JAL programs to pure Java programs, and a library of supporting classes, called the Jack Agent Kernel.

Analysis There is no evidence about any analysis method.

Design It is assumed that the definition of the agents' functionality has been provided, and that the BDI model has been chosen. Then, this stage consists of the identification of the elementary classes required to manipulate the domain objects (perceptions, actions, domain-specific data structures), and the identification of the mental states elements of the agents (interactions, goals, beliefs, plans). This applies only if you choose the proposed Jack's BDI agent model. In this case, a good knowledge of the Jack BDI model is required. The lack of a real design method makes reuse difficult.

Development Development is a combination of normal Java coding for the elementary classes and extended Java (Jack Agent Language) for the agent-specific components. The extensions allow describing agent's behaviour, capability, plans, events and relational database into Java code. The Jack Agent Compiler then translates this extended Java into pure Java. A graphical software tool is also provided to facilitate the management of large projects. The Jack development stage is applicable to multi-agent systems composed of BDI agents, possibly interfaced with legacy systems. Note that Jack did not impose to use its own BDI model, and that other agent models can be used (but have to be implemented), while benefiting of the Jack Agent Kernel services. Both a Java programming background, and a Jack's BDI engine knowledge is required. The modularity of Jack enables a good reuse of code.

Deployment No deployment tool is provided. Deployment consists of manually launching the agent's classes.

Other At the time of writing this paper, Jack is freely available under a 60-day evaluation license. Permanent or commercial licenses are at the authors' discretion. Jack is still under evolution, and Agent Oriented Software proposes many commercial services.

Comments Jack is particular by its strong agent-programming orientation. This leads to a high versatility, agent's architecture can range from simple Java-coded reactive behaviours to full BDI, using the provided architecture or another one. Unfortunately, the documentation provided is very technical, and does not cover the methodological aspects, especially for the analysis and the design. The deployment also lacks of support. The reason for that is maybe that Jack seems to be mainly used as an internal tool by Agent Oriented Software Pty. Ltd. to provide its services.

MadKit (`http://www.madkit.org/`)

MadKit is a Java multi-agent platform built upon an organisational model. It is developed by Olivier Gutknecht and Jacques Ferber at the LIRMM (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier), a public research laboratory in France. Contrarily to the other platforms presented in this paper, MadKit is mostly a MAS runtime engine, using an agent micro-kernel. The underlying organisational model is named Aalaadin [8]. A short methodology is proposed in [9].

Analysis No specific analysis method is associated with MadKit. This stage should include a functional analysis, a dependency analysis, a group context discovery and a choice of the coordination mechanisms. Any domain-specific or generic analysis method can be used.

Design Since MadKit is mainly organisation-oriented, this stage includes the definition of the organisational model (groups, roles), the interaction model (protocols, messages), and other specific entities (tasks, goals, etc). The Aalaadin model serves as a guideline for the design, but no software tools are provided. Despite the organisation orientation of the Aalaadin model, it is in fact applicable to a broad range of MAS designs. The completion of this stage is facilitated by the intuitiveness of groups and roles formations. Additionally, group and roles definitions can be reused, for example through design patterns.

Development This stage includes the choice of the agent model, its implementation, and the implementation of interaction protocol strategies. No agent model is provided, it has to be implemented in Java from scratch, or modified to work with MadKit. Since no assumption is made about the agent model, any kind of model can be used (but preferably simple ones). But as all the agent code has to be implemented in Java, it can be a heavy task to develop complex cognitive agents. Luckily, agent models implemented for MadKit can be reused across projects.

Deployment The deployment of MadKit agents takes place in the G-box, a kind of agent's sandbox, where agents can be created, modified and destroyed, with a nice graphical interface. Several G-boxes can be connected to achieve distribution across a network. A simple console mode is also available to simplify deployment. The G-box allows dynamic configuration of agent's editable features, in a similar way of JavaBeans. No global profiling is available. Agents are packaged into Jar archive, and can be mixed with other agents, allowing a high level of agent's reusability.

Other At the time of writing this paper, MadKit is free for educational use. Commercial use is at the discretion of the authors. MadKit is still under evolution, in order to add new features and to fix bugs.

Comments The main characteristic of MadKit is that it is mostly a multi-agent runtime engine. This leads to simple development and deployment, since the platform focuses on agents' infrastructure. The lack of a methodology (apart from the Aalaadin model, that is more a descriptive organisational model than a conception method) is in way of being overcomed by recent works [9], but still needs to be completed. The main default of MadKit, is that building complex agents requires writing a lot of code, since there is no pre-established agent model. On the good side, it adds flexibility, and any programmer can begin to write its own agents, even without a good multi-agent systems background, making MadKit very interesting in an educational context.

Zeus (`http://www.labs.bt.com/projects/agents/zeus/`)

Zeus is an integrated environment for the rapid building of collaborative agents applications. It is developed by the Agent Research Programme of the British Telecom Intelligent System Research laboratory. The Zeus documentation is abundant, and puts a strong emphasis on the importance of the methodological aspect of Zeus ('*The agent creation methodology is vital to the use of the Zeus toolkit*' [10]). The Zeus methodology uses the same four-stage decomposition for agent development as in our analysis, respectively domain analysis, design, realisation and runtime support.

Analysis The analysis stage, which consists of role modelling, is described in [11]. At this stage, no software tool is provided. As agents are defined by their role and by their behaviour, this stage is specifically applicable to role-oriented, rational multi-agent systems. The modelling of the roles is done with UML class diagrams and patterns, thus avoiding new formalisms and accessible to a broad audience. On the reuse aspect, a large number of commons role models are provided, covering information management, trading and business processes.

Design The agent design stage, which consists of finding solutions that fulfil the role responsibilities defined in the previous stage, is supported by three cases studies [11]: FruitMarket (Trading), PC Manufacture (Supply chain) and Maze Navigator (Rule based agents). There exists a software tool at this stage. The underlying Zeus agent model limits the design to task-oriented, goal-driven, collaborative agents. Since the process of finding a solution to a given problem is the most difficult thing to systematise, the design stage mainly requires design skills, but is not technically difficult. The three case studies are a good starting point for reuse. The simplicity of the design formalism used by Zeus facilitates design reuse, but lacks of formalism (mainly composed of natural language statements).

Development The agent development stage is both covered by the three supplied case studies and the Application Realisation Guide [11]. The five activities involved for developing agents are both supported by graphical software tools These five activities are: Ontology Creation, Agent Creation, Utility Agent Creation, Task Agent Configuration and Agent Implementation. These activities require a good knowledge of the tools, which are hopefully well documented. Of course this stage is only applicable to the Zeus agent model. Ontologies can be easily reused. There is still no support for agent reuse across projects. The general modularity of the development tool enables the reuse of frequently used agent functionalities, but adding a new functionality requires to return to the (Java) code (for example, to add a new agent co-ordination strategy).

Deployment The deployment stage is documented in the Runtime Guide [11]. This document describes how to launch the generated MAS, and how to use the Visualiser tool. There are also some considerations about the art of debugging. The visualiser tools visualises the MAS from different points of view: organisation and interactions in the society, global task decomposition, statistics and internal states of agents. It is also possible to control any individual agent states, and even to configure agents at run-time. The deployment tools use user-friendly graphic interfaces that facilitate the deployment. Modifications to the multi-agent systems' society composition or localisation require recompilation, thus making the deployment reuse next to impossible.

Other The complete Zeus toolkit is available for download on the Zeus web site. Zeus is Open Source software, hence is free of charge for academic and industrial users. No official support is provided. A Zeus mailing list is available.

Comments The main particularity of Zeus is its complete integration of all the stages from design to deployment. It provides theoretical and practical tools, uses actual techniques of software engineering (design patterns, UML), and its methodological documents focus on the *how to* and not only on the *what to do*. However, even if Zeus is modular, there is only one agent model supported, which limits the range of possible designs of multi-agent systems. Also, as any complex tool, Zeus is long and difficult to master, but with the benefice of a great productivity once it is mastered.

5 Discussion

As we have seen with the four platforms presented here, the growing interest in multi-agent development platforms has lead to very interesting tools. Of course this is only a comparison between some specific platforms, and since we believe there is no absolute MAS methodology and platform, any comparison has some bias. The following table illustrates the qualitative values of the several criteria (Completeness, Applicability, Complexity, Reusability) we have used to analyse the four platforms (AgentBuilder, Jack, MadKit, Zeus).

Stage	Analysis				Design				Development				Deployment				Su
Quality	C	A	C	R	C	A	C	R	C	A	C	R	C	A	C	R	
AgentBuilder	■	■	■	■	■	■	▨	▨	■	▨	▨	■	■	▨	■	□	▨
Jack	□	□	□	□	▨	▨	▨	□	■	■	■	■	□	□	□	□	▨
MadKit	□	□	□	□	▨	■	■	■	▨	▨	▨	■	■	■	■	■	■
Zeus	■	■	■	■	■	■	■	■	■	▨	▨	▨	■	■	■	□	■

Good ■ Average ▨ None □

From the methodological point of view, we observe that there exist big differences between platforms, from the Zeus's *vital* methodology, to the quasi-absence of methodological material or tools. Often, the starting point of a platform is an *implementation* tool, but such a tool is not self-sufficient. The programmer needs a manual to use it. A descriptive manual is not enough, because the fundamental question a programmer asks himself is "How do I build a multi-agent system from that specification?" and this question is not answered by a simple description of the tool. The methodological documentation of a platform must help the programmer, by defining simple tasks that, step by step, lead from the analysis to the deployment of his application.

From the technical point of view, different solutions have been chosen: AgentBuilder and Zeus are more like "BDI editors", whilst Jack is an agent programming language and MadKit a multi-agent runtime infrastructure. Each kind of solution has its advantages and drawbacks. BDI editors are very efficient because they allow programming in a high level of abstraction, but are committed to single agent architecture. On the other hand, agent programming languages are more versatile, but at the expense of more code writing, because the level of abstraction is lower. A better solution would be a modular platform, that would give to the programmer a rich library of agent models, where each model would have its own dedicated editor. This would combine good versatility and high-level programming, even at the price of a higher platform development cost.

We have also noted all these platforms lacks some important aspects of multi-agent systems, such as the notion of environment, which can play a very important role in some applications [12], and have difficulties to apprehend the agent society as a whole. Coordination and cooperation aspects are at best limited to the definition of groups and the design of interaction protocols. The engineering of societies, especially emergent organisations [13], continue to show too much conceptual, methodological and technical difficulties to become a standard of multi-agent platforms.

We are currently working in the Magma group on modular multi-agent platforms that address as much as possible the proposed criteria : the MASK [14] platform and its successor, Volcano, currently under development. In these platforms, multi-agent systems are composed of four parts: Agent, Environment, Interaction and Organisation, corresponding to the Vowels methodology [15]. Each part is represented in the platform as a toolbox, containing a library of models from which the developer can choose the most adequate one and then instanciate it. We are also developing the accompanying methodology, to cover as much as possible the four stages. This should result into a revised platform in some future, which will better reflect our Vowels methodology [15].

References

1. Federico Bergenti and Agostino Poggi. Exploiting UML in the Design of Multi-Agent Systems. In this volume.
2. Y. Shoham. AGENT-0: a simple agent language and its interpreter. *In Proceedings of the Ninth National Conference on Artificial Intelligence*, Vol II (pp. 704-709), Anaheim, CA, MIT Press, 1991.
3. Y. Shoham. Agent Oriented Programming. Artificial Intelligence, 60(1), pp. 51-92, North-Holland, 1993.
4. S. R. Thomas. PLACA, an Agent Oriented Programming Language. Ph.D. Thesis, Stanford University, 1993.
5. Reticular Systems. AgentBuilder User's guide. External documentation, `http://www.agentbuilder.com/`, August 1999.
6. Mark d'Inverno, David Kinny, Michael Luck, and Michael Wooldridge. A formal Specification of dMARS. In Singh et al, editors, *Proceedings of the 4th International Workshop on Agent Theories, Architectures, and Languages (ATAL'97)*, LNAI, Vol. 1365, pp. 155-176, Springer, 1998.
7. P. Busetta, R. Rönnquist, A. Hodgson, A. Lucas. JACK Intelligent Agents – Components for Intelligent Agents in Java. Updated from AgentLink Newsletter #2, `http://www.agent-software.com.au/`, October 1999.
8. J. Ferber and O. Gutknecht. A meta-model for analysis and design of multi-agent systems. *Proceedings of the 3rd International Conference on Multi-Agent Systems*, (ICMAS'98), IEEE, pp. 155-176, August 1998.
9. O. Gutknecht and J. Ferber. Vers une méthodologie organisationnelle pour les systèmes multi-agents. In *Actes des JFIADSMA'99 (Journées Francophones d'Intelligence Artificielle Distribuée et Systèmes Multi-Agents)*, Saint-Denis, Reunion, 1999
10. J. Collis, D. Ndumu. The ZEUS Technical Manual. external documentation, `http://www.labs.bt.com/projects/agents/zeus/`, September 1999.
11. J. Collis, D. Ndumu, and S. Thompson. ZEUS Methodology Documentation, Role Modelling Guide, Three case studies, Application Realisation Guide, Runtime Guide. `http://www.labs.bt.com/projects/agents/zeus/`, August-November 1999.
12. H. Van Dyke Parunak, Sven Brueckner, John Sauter and Robert S. Matthews. Distinguishing Environmental and Agent Dynamics: A Case Study in Abstraction and Alternate Modeling Technologies. In this volume.
13. Cristiano Castelfranchi. Engineering Social Order. In this volume.
14. M. Occello, C. Baeijs, Y. Demazeau, and J.L. Koning. MASK : An AEIO Toolbox to Develop Multi-Agent Systems. In Cuena et al editors, *Knowledge Engineering and Agent Technology, IOS Series on Frontiers in Artificial Intelligence and Applications*, 2000.
15. Y. Demazeau, M. Occello, C. Baeijs, and P.-M. Ricordel. Systems Development as Societies of Agents. In Cuena et al editors, *Knowledge Engineering and Agent Technology, IOS Series on Frontiers in Artificial Intelligence and Applications*, 2000.