A Formal Specification of M-Agent Architecture

Krzysztof Cetnarowicz[1], Pablo Gruer[2], Vincent Hilaire[2], and Abder Koukam[2]

[1] Institute of Computer Science
AGH – University of Mining and Metallurgy
Al. Mickiewicza 30, 30-059 Krakow, Poland
[2] UTBM/Systèmes et Transports
Belfort Technopôle
90000 Belfort, France

Abstract. Complexity of distributed and decentralized systems demands new tools for designing and programming processes. An idea of autonomous agents that arises as an extension of the object and process concepts may be applied to distributed and decentralized systems development. In the paper the authors have undertaken an attempt to describe formally the architecture of multiagent systems using a method of specification based upon the combination of Object-Z, Statecharts and M-agent architecture. The proposed method of multiagent system description may be considered as a starting point to develop a multi-agent system description method covering a gap existing between theoretical analysis and practical realization of multiagent systems.

1 Introduction

Complexity of distributed and decentralized systems demands new tools for designing and programming processes. An idea of autonomous agents that arises as an extension of the object and process concepts may be applied to distributed and decentralized systems development ([4],[5]). The approaches proposed, (BDI architecture, Agent-0, etc.) ([14], [17]), ([15]) make some attempts to a practical model of multiagent system definition, but it seems to the authors that these attempts are not always practically applicable, and particularly they present, among others, the following inconveniences: They take as a base of considerations a low level of abstraction of the problem, and therefore the proposed model of the agent architecture does not cover a wide range of types of agents (or multiagent systems). The formalism proposed is not always convenient in practical applications where the real problem with its characteristic features is to be interpreted in the formal model. The approaches proposed do not take into consideration such problems as testing and debugging of the creating system. It seems that the research work is grouped around two opposite points: - one representing very formal approach that enables to describe some properties of multiagent systems with formal methods, - the other representing a practical approach with tools that enables to develop practical multiagent systems without any control of a number of important properties (such as stability, efficiency, etc.) of the created multiagent systems. In order to provide precise, unambiguous meanings for the concepts and terms introduced previously, we introduce a formal modelling approach. This formal modelling approach is based upon the combination of Object-Z [6] and Statecharts [9]. In doing so, we can specify many aspects of the MAS that may be verified with theoretical tools and then applied in a practical way. The proposed method of a multiagent system description and modelling may be considered as a starting point to develop a multi-agent system description method covering a gap existing between a theoretical analysis and practical realization of multiagent systems.

The rest of this paper is organized as follows. The M-agent approach and its formal framework are presented in section 2. To illustrate the proposed methodology, a decentralized system of medical help is presented in section 3. Finally, the last section includes a summary and an outline of future research.

2 Formal Framework for M-Agent

2.1 Principle of the M-Agent Architecture

The M-Agent architecture [3]) is based upon the split of MAS in agents and an environment. The environment is what is observed by the agents and represented by a corresponding model. Each agent is related to a subset of the environment which is pertinent to it. The whole reality analyzed from the point of view of the multiagent system may be divided into two parts (fig. 1): *environment* - that may be observed by the agent and represented by a corresponding model, *agent′s mind* - an area in which the agent builds and processes a model of the environment.

We can consider that a given agent *a* remains and acts in the environment, and to realize thit for any agent the following notions are defined:

- *models* of the environment, *strategies*, and *goals*.
- operation of the environment observation - *Imagination* and operation of the selected strategy execution *Execute*.

Every agent is characterized by an agent's mind. This mind is composed of strategies, goals and models of the environment. Strategies are what agents can do for modifying their models. Goals are what agents try to satisfy. The main idea of the agent functioning is the following: The agent observes the environment around and builds a *model* of the environment in its mind. For this purpose it uses the *Imagination* operation. The agent forecasts its possibilities using one of the available *strategies*. Applying a given

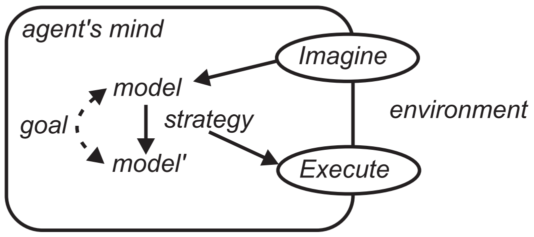


Fig. 1. Principle of the M-agent architecture.

strategy to the *model* of the environment, it obtains the forecasted *model* of the modified environment. Then the agent compares in its mind the *models* (real and forecasted) using the *goal* function that determines the goal of agent functioning. The agent selects the best strategy available according to its goals and realizes it. If the evaluation of the forecasted model is satisfactory, the agent realizes the selected strategy which is represented by the *Execution* operation. The whole decision process is carried out using the models of the environment processing in the agent's mind. In some cases we may observe that beings alter their behaviour as if they were changing their point of view. In this case the agent may be described with a more complex M-agent model called a multiprofile M-agent. Using the multiprofile M-agent architecture we can decompose the agent specification into several *profiles* (fig. 2). The operations of the *strategy execution* is common for all profiles, but the models, strategies, goals and imagination functions are different and characteristic for each profile. Every profile works in the following way:

As a consequence the agent obtains the *optimal strategy* proper for each profile. Then using the common decision function the agent selects the best strategy for the execution.

The final decision to realize the *strategy* is made by the agent with the use of the *Decision* function that takes into consideration the *models* of all the profiles.

The MAS environment is specified by the *Environment* which may take a form of a class. This class is composed of a set of resources, set of agents and topology defining the spatial properties of the environment.

2.2 Formal Framework

We build a framework written with formalism which is a composition of Object-Z and statecharts. The whole composition mechanism is described in [11]. For the sake of clarity this mechanism will only be sketched. In order to compose Object-Z classes and statecharts one can write a class with a schema including a statechart. This statechart specifies the behaviour of the class. Our framework is composed of a set of such classes which specify all meta-model concepts: environment, profile, and agent.

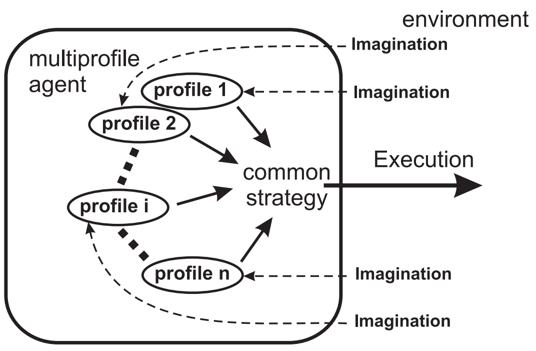


Fig. 2. Principle of the multiprofile M-agent architecture.

The structure of the framework as a class hierarchy allows for inheritance from defined classes in order to define more specific model concepts or to extend existing concepts. Every model concept is given formal semantics by a class of the framework.
The first concept of the model is a profile. The profile is defined by a Model of the environment, a set of actions and a set of goals to be reached. The *behaviour* of sub-schema specifies the behaviour of the profile. This schema is to be refined by including a statechart.

Profile

$$\begin{array}{l}
\mathit{Model} == \mathbb{P}\,\mathit{Attribut} \\
\mathit{Strategy} == \mathbb{P}\,\mathit{Action} \\
\mathit{Goal} == \mathbb{P}\,\mathit{Predicate}
\end{array}$$

behavior

$$\mathit{mind} : \mathit{Model} \times \mathit{Strategy} \times \mathit{Goal}$$

Environment

$$\begin{array}{l}
\mathit{agents} : \mathbb{P}\,M - \mathit{Agent} \\
\mathit{stimulus} : \mathbb{P}\,\mathit{Event} \\
\mathit{reactions} : \mathbb{P}\,\mathit{Event}
\end{array}$$

reactionsComputation

$$\begin{array}{l}
\Delta \mathit{reactions} \\
\mathit{reaction}' \\
\hline
\mathit{reactions}' = \bigoplus_{\forall\, i \in \mathit{stimulus}} i
\end{array}$$

$$\begin{array}{l}
\square(\forall\, a \in \mathit{agents}, e : \mathit{Event} \bullet \\
\qquad e \in a.\mathit{pertinentEvents} \Rightarrow \\
\qquad \bigcirc(\exists\, d \in a. \bullet d \in \mathit{reactions} \\
\qquad\qquad\qquad \wedge\ d \in \mathit{reaction}'!)
\end{array}$$

This work deals with situated MAS, that is to say, the MAS in which interactions occur through the environment. Situated MAS environments may have a metric, that is to say a total mapping from the entity which relates it to places. Here we do not assume the existence of such a mapping. The environment groups all agents of the MAS and is the place where all interactions are combined. The MAS environment is specified by the *Environment* class. This class is composed of a set of agents which are the MAS agents. The *stimulus* set groups all events occurring at a moment in time. The *reaction* set is the set of all agent reactions. Reactions are calculated by the reactionsComputation operation. This operation is obviously to be refined to make the combination operator (circled plus) at least more precise.

2.3 M-Agent Specification

In order to specify M-Agents the $M - \mathit{Agent}$ class which specifies a generic agent based upon the M-Agent architecture is added to the agent framework, defined in [12]. A specific M-Agent is then defined as a set of *Event* it reacts to, *pertinentEvents*. The reaction depends on the active agent's mind. The agent's mind defines values of models, strategies and goals. The agent's mind model and strategies are defined on the basis of the agent profiles. Indeed, $\mathit{agentMind}.\mathit{Model}$ is a subset of the agent's attributes and $\mathit{agentMind}.\mathit{Strategy}$ is a subset of the agent's actions which are an union of profile attributes and profile actions respectively. These profiles define the attributes, actions and stimulus of the M-Agent. The temporal invariant of the class constrains the behavior of the M-Agent so that they react to whatever event belonging to *pertinentEvents*. The reaction consists in applying the best strategy available with respect to the goals of the M-Agent. Intuitively, the best strategy is the strategy which satisfies the maximum number of sub-goals.

$$
\begin{array}{l}
\textit{M} - \textit{Agent} \\
\hline
\quad \textit{attributes} : \mathbb{P}\,\textit{Attribute} \\
\quad \textit{actions} : \mathbb{P}\,\textit{Action} \\
\quad \textit{stimulus} : \mathbb{P}\,\textit{Event} \\
\quad \textit{pertinentEvents} : \mathbb{P}\,\textit{Event} \\
\quad \textit{activeProfiles} : \mathbb{P}\,\textit{Profile} \\
\quad \textit{agentMind} : \textit{Model} \times \textit{Strategy} \times \textit{Goal} \\
\hline
\quad \textit{agentMind} = \cup_{\forall\, p \in \textit{activeProfiles}} p.\textit{mind} \\
\quad \forall\, m \in \textit{agentMind}.\textit{Model} \bullet m \subseteq \textit{attributes} \\
\quad \forall\, s \in \textit{agentMind}.\textit{Strategy} \bullet s \subseteq \textit{actions} \\
\hline
\Box(\exists\, e : \textit{Event}, m : \textit{Model}, s : \textit{Strategy} \bullet \\
\qquad (e\ \textbf{occurs} \wedge e \in \textit{pertinentEvents} \wedge m = \textit{agentMind}.\textit{Model}) \\
\qquad \Rightarrow \bigcirc(\textit{agentMind}.\textit{Model} = s(m) \wedge s \in \textit{agentMind}.\textit{Strategy} \wedge \forall\, s' : \textit{agentMind}.\textit{Strategy} \\
\qquad \bullet\ s \neq s' \\
\qquad \bullet\ d : \textit{Predicate}/s(m) \vdash d, \\
\qquad\ \ d' : \textit{Predicate}/s'(m) \vdash d' \\
\qquad \Rightarrow (\textit{agentMind}.\textit{Goal} \Rightarrow d) \wedge (d \Rightarrow d')) \\
\hline
\end{array}
$$

3 Example

3.1 The Medical Help System

The objective of the Medical Help System (MHS) consists in selecting the best adapted hospital, capable of providing medical assistance to a patient. The MHS could ideally help to direct, in real time, an ambulance with the victim of some kind of health hazard (accident, stroke, ...) to the appropriate care unit.

The system is to be implemented as a MAS, in which ultimately, active entities will be mobile agents with information about symptoms of the given patient. A hospital is represented by agents (*HospitalAgents*) and constitutes the node of the network, and a patient is represented by *PatientAgent*. Each *PatientAgent* moves in the network and holds negotiations with *HospitalAgent*. The concept of the system structure and the principles of cooperation of each agent is reduced to the following postulates: Hospital (wards) is represented as nodes in the network. The possibilities of a hospital in the given node are represented by a *HospitalAgent*. The patient (sick, injured person) is represented by a *PatientAgent*. It has information about the examination results/treatment and the present symptoms of the given patient. In the system there are nodes authorized to create the *PatientAgent*. These nodes are accessible to general practitioners and emergency services (e.g. firemen, police, etc.) by means of overhead or radio network. In the system there is a unified multiagent platform which enables *PatientAgents* to move around in the network and hold negotiations between them and *HospitalAgents* concerning a possibility of providing help and parameters (conditions) of the service. In the system there are (can be

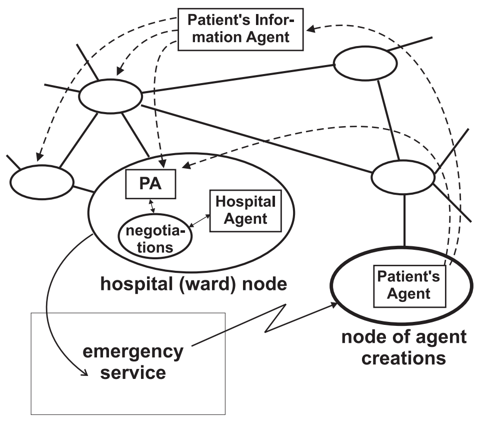


Fig. 3. Scheme of actions of the decentralized multiagent system of providing medical help.

generated) *PatientInformationAgents* which look for information about the given patient in medical centers (outpatient clinics, hospitals) to provide it to *PatientAgent* in order to create the patient's full picture (epicrisis) enlarged by dispersed archival data. In fact, this example has yet been developed in [2]. This paper introduces a retro-engineering approach for this example.

3.2 Profile Specification for the MHS

The specification process begins by identifying the profiles and their interactions. Each agent will encapsulate some profile. Indeed, *Patient* agents will search a hospital, contact an emergency service which will be in relationship with all hospitals services. In this paper, *HospitalSearch* and *AdmissionNegotiation* profiles will be described.

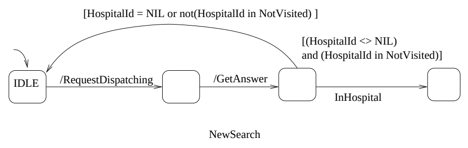
Let us consider now the formal specification of the *HospitalSearch* profile, i.e. Object-Z class *HospitalSearch*. Basic types $[HospitalId, Symptom]$ represent hospital identifiers and medical information about the patient respectively.

HospitalSearch

Profile

$symptoms : \mathbb{P}\, Symptom$
$notVisited : \mathbb{P}\, HospitalId$
$hospitalId : HospitalId$

Behavior


[HospitalId = NIL or not(HospitalId in NotVisited)]
[(HospitalId <> NIL)
and (HospitalId in NotVisited)]
IDLE
/RequestDispatching
/GetAnswer
InHospital
NewSearch

INIT

$visited, notVisited = HospitalId$

RequestDispatching

$\Xi(HospitalSearch)$
$description! : \mathbb{P}\, Symptom$

$description! = symptoms$

GetAnswer

$\Delta(HospitalSearch)$
$identification? : HospitalId \cup \{NIL\}$

$hospitalId' = identification?$
$identification? \neq NIL$
$\quad \Rightarrow notVisited' = notVisited \setminus \{identification?\}$
$\qquad \wedge\ visited' = visited \cup \{identification?\}$
$identification? = NIL$
$\quad \Rightarrow notVisited' = notVisited$
$\qquad \wedge\ visited' = visited$

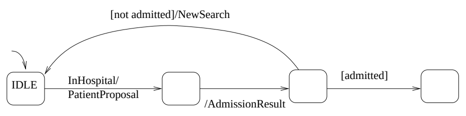
The *HospitalSearch* class includes two operations, *RequestDispatching* and *GetAnswer*. The first operation interacts with the *Dispatching* profile of some *Hospital* agent. In fact, this operation which does not modify the profile states (Ξ notation) outputs *description*!, a list of symptoms. The postfixed notation ! denotes outputs values. As a result of the *GetAnswer* operation, the patient obtains an identifier of the hospital with which negotiations for admission are conducted. This negotiation operation modifies the profile state (Δ notation) and takes as input, ? postfix notation, a hospital identifier. The behaviour defined by *HospitalSearch* is to execute first *RequestDispatching* and then *GetAnswer*. If there are no hospitals available or the one found has already been visited then a new search is started. In the case of a new hospital *InHospital* event is broadcasted.
The admission negotiation profile class is as follows:

AdmissionNegotiation

Profile
$InHospital, NewSearch : Event$

$hospitalId : HospitalId$
$symptoms : \mathbb{P}\, Symptom$
$rights : InsuranceProfile$
$availableCash : \mathbb{R}^+$
$admitted : \mathbb{B}$

Behavior



PatientProposal

$\Xi(AdmissionNegotiation)$
$description! : \mathbb{P}\, Symptom$
$rights! : Insurance$

$description! = symptoms$
$rights! = rights$

AdmissionResult

$\Delta(AdmissionNegotiation)$
$cure? : Cure$
$price? : \mathbb{R}^+$

$cure? = \varnothing \Rightarrow \neg\ admitted'$
$price? \geq availableCash \Rightarrow \neg\ admitted'$
$cure? \neq \varnothing \wedge price? \leq availableCash \Rightarrow admitted'$

Patient

$M - Agent$

$Profile$
$= \{HospitalSearch, AdmissionNegotiation\}$
$pertinentEvents = \{InHospital, NewSearch\}$
$mind$
$\quad = \{hospitalSearch, AdmissionNegotiation\}$
$hospitalSearch.Model == (visited, notVisited)$
$hospitalSearch.Strategy == getAnswer$
$hospitalSearch.Goal == InHospital$ **occurs**
$AdmissionNegotiation.Model == admitted$
$AdmissionNegotiation.Strategy$
$\quad == AdmissionResult$
$AdmissionNegotiation.Goal == admitted$

INIT

$agentMind = hospitalSearch$

$InHospital$ **occurs**
$\quad \Rightarrow \bigcirc(mind = AdmissionNegotiation)$

$NewSearch$ **occurs**
$\quad \Rightarrow \bigcirc(mind = hospitalSearch)$

The behaviour of *AdmissionNegotiation* consists in waiting for *InHospital* event, then executing *PatientProposal* and then carrying out *AdmissionResult* operations. If not admitted then *NewSearch* event is broadcasted. *PatientProposal* operation outputs a list of symptoms and details on the insurance of the patient. *AdmissionResult* operation decides if treatment is compatible with patient symptoms and available cash. Both profiles communicate by the intermediate of events *InHospital* and *NewSearch*.

3.3 Agent Specification for the MHS

The *Patient* class specifies a specific M-Agent of the MAS for Medical Help. This agent proposes *HospitalSearch* and *AdmissionNegotiation* profiles. The pertinent events for this agent are *InHospital* and *NewSearch*. The mental states available are *hospitalSearch* and *admissionNegotiation*. The first active mental state of a *Patient* is *hospitalSearch*. The two temporal invariants specify that each time a *Patient* is in hospital (resp. begins a search for a hospital) *admissionNegotiation* (resp. *hospitalSearch*) becomes the active mental state. The model for *hospitalSearch* mind consists in *visited* and *notVisited* hospitals. Each time a new hospital is visited its identifier is added to *visited*. The *goal* is to enter a hospital which happens when *InHospital* occurs. The model for *admissionNegotiation* mind is Boolean *admitted*. The goal here is to be admitted to hospital, which happens when *admitted* is true.

4 Conclusion

The language used by the specification framework can describe reactive and functional aspects. It is structured as a class hierarchy so one can get inheritance from these classes to produce its own specification. As an example, we have specified the M-Agent specific agent architecture by adding a new class to the framework. Several MAS have already been specified with this framework without using a specific agent architecture [10,12]. These MAS have been applied to the radio-mobile network field and foot-and-mouth disease simulation.
The specification language allows prototyping of specification [12]. Prototyping is not the only means of analysis, indeed, in another work [7], we have introduced a formal verification approach. Moreover, the specification structure enables incremental and modular validation and verification through its decomposition. Eventually, such a specification can be refined to an implementation with multi-agent development platform like MadKit [8] which is based upon an organizational model.
Formal theories are numerous in the MAS area but they are not all related to concrete computational models. Temporal modal logic, for example, have been widely used [19]. Despite an important contribution of this work to a solid underlying foundation for MAS, no methodological guidelines are provided concerning the specification process and how an implementation can be derived. Another type of approach consists in using traditional software engineering or knowledge based formalisms [16]. Among these approaches a few [16,1] provide the specifier with constructs for MAS decomposition. However, there are no proposed methodologies for these approaches. In [13] there is a survey of MAS methodologies. MaSE is more detailed in terms of guidance for system designer but

necessitates several differents models. Gaia is based upon a logical formalism which is difficult to refine down to an implementation.
The presented approach of coupling the M-agent architecture with Object-Z and State-chart formalism gives a homogenous system for the description of the multiagent systems. Taking as a starting-point a properly defined description of the problem we may developed a base for the Agent Oriented Analysis, Design and even Programming and use the specified model of a given problem to go toward formal verification or practical realization.

References

1. F.M.T. Brazier, B. Dunin Kęplicz, N. Jennings, and J. Treur. Desire: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6:67–94, 1997.
2. K. Cetnarowicz and E. Cetnarowicz. Multi-agent decentralised system of medical help. In *Management and Control of Production and Logistics. IFIP, IFAC, IEEE Conference*, Grenoble, France, 2000. ENSIEG, LAG Grenoble, France 2000.
3. K. Cetnarowicz and E. Nawarecki. Système d'exploitation decentralisé realisé à l'aide de systèmes multi-agents. In *Troisième Journées Francophone sur l'Intelligence Artificielle Distribuée et les Systèmes Multiagents*, pages 311–322, St Baldoph, Savoie, Francja, 1995.
4. Y. Demazeau and J.-P. Müller. Decentralized artificial intelligence. In Y. Demazeau and J. P. Müller, editors, *Decentralized A.I.*, pages 3–14. North-Holland ISBN 0-444-88705-9, 1990.
5. Y. Demazeau and J. P. Müller. From reactive to intentional agents. In Y. Demazeau and J. P. Müller, editors, *Decentralized A.I. 2*, pages 3–10. North-Holland, 1991.
6. Roger Duke, Gordos Rose, and Graeme Smith. Object-z: Specification language advocated for the description of standards. Tech. rep. no. 94-95, Software Verification Research Centre, Dept. of Computer Science, the University of Queensland, Quinsland, Australia, 1994.
7. Pablo Gruer, Vincent Hilaire, and Abder Koukam. an Approach to the Verification of Multi-Agent Systems. In *International Conference on Multi Agent Systems*. IEEE Computer Society Press, 2000.
8. Olivier Gutknecht and Jacques Ferber. The madkit agent platform architecture. In *1st Workshop on Infrastructure for Scalable Multi-Agent Systems*, june 2000.
9. David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
10. V. Hilaire, T. Lissajoux, and A. Koukam. Towards an executable specification of Multi-Agent Systems. In Kluwer Academic Publisher, editor, *International Conference on Enterprise Information Systems'99*, 1999.
11. Vincent Hilaire. *Vers une approche de spécification, de prototypage et de vérification de Systèmes Multi-Agents*. PhD thesis, UTBM, 2000.
12. Vincent Hilaire, Abder Koukam, Pablo Gruer, and Jean-Pierre Müller. Formal specification and prototyping of multi-agent systems. In *Engineering Societies in the Agents'World*, number 1972 in Lecture Notes in Artificial Intelligence. Springer Verlag, 2000.
13. Carlos Iglesias, Mercedes Garrijo, and José Gonzalez. A survey of agent-oriented methodologies. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555 of *LNAI*, pages 317–330, Berlin, July 04–07 1999. Springer.
14. D. Kinny, M. Georgeff, and A. Rao. A methodology and modelling technique for systems of bdi agents. In Van Velde and Perram [18], pages 56–71.

15. Crowley J. L. and Demazeau Y. Principles and techniques for sensor data fusion. In *Signal Processing*, volume 32, pages 5–27, Elsevier Science Publishers B. V., 1993.
16. Michael Luck and Mark d'Inverno. A formal framework for agency and autonomy. In AAAI Press/MIT Press, editor, *Proceedings of the First International Conference on Multi-Agent Systems*, pages 254–260, 1995.
17. A. Rao. Agentspeak(l): Bdi agents speak out in a logical computable language. In Van Velde and Perram [18], pages 42–55.
18. W. Van Velde and J. W. Perram, editors. *7th European Workshop on Modelling Autonomous Agents in Multi-Agent World, MAAMAW'96*. Number 1038 in Lecture Notes in Artificial Intelligence. Springer-Verlag ISBN 3-540-60852-4, Berlin, 1996.
19. Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. Available by FTP, 1994. Submitted to The Knowledge Engineering Review, 1995.