5-2007

An Intelligence-Aware Process Calculus for Multi-Agent System Modeling

Raheel Ahmad
Southern Illinois University Carbondale

Shahram Rahimi
Southern Illinois University Carbondale, rahimi@cs.siu.edu

Bidyut Gupta
Southern Illinois University Carbondale

Published in Ahmad, R., Rahimi, S., & Gupta, B. (2007). An intelligence-aware process calculus for multi-agent system modeling. International Conference on Integration of Knowledge Intensive Multi-Agent Systems, 2007. KIMAS 2007, 210-215. doi: 10.1109/KIMAS.2007.369811

An Intelligence-Aware Process Calculus for Multi-Agent System Modeling

Raheel Ahmad, Shahram Rahimi, Bidyut Gupta
Department of Computer Science,
Southern Illinois University,
Carbondale, IL - 62901
{rahmad, rahimi, bidyut}@cs.siu.edu

Abstract— *In this paper we propose an agent modeling language named CAML that provides a comprehensive framework for representing all relevant aspects of a multi-agent system: specially, its configuration and the reasoning abilities of its constituent agents. The configuration modeling aspect of the language supports natural grouping and mobility, and the reasoning framework is inspired by an extension of the popular BDI theory of modeling cognitive skills of agents. We present the motivation behind the development of the language, its syntax, and an informal semantics.*

1. INTRODUCTION

Multi-Agent Systems have appeared as a crucial and exciting field in computer science in the last couple of decades. A relevant and popular definition of an agent is given by Wooldridge [17]: "An agent is a computer system situated in some environment and that is capable of flexible, autonomous action and communication with other agents in this environment in order to meet its design objectives." An agent is often specified by its autonomous nature, communication capabilities, its location in an environment, and its ability to react to and affect change in its environment. A multi-agent system then is a recognized collection of such agents that coexist and interact in an *environment* where the overall control and information is generally distributed (decentralized). Multi-agent systems find application in a wide range of domains including: business and finance including e-commerce, human-computer interactions, information management, traffic control, social sciences, computer games, and several Internet-based applications such as search agents.

The ad-hoc nature of development, analysis and verification of today's multi-agent systems has resulted in largely unreliable products for the end-user. Formal methods are almost a necessity when it comes to developing systems in critical scenarios such as military, medical systems, and air traffic control [1, 2]. Although they represent a significant investment in time and expertise, the advantages are significant: *formal specification of* a system by itself can be useful in terms of getting the system requirements right by eliminating errors at the design phase; *verification* of a system can allow designers to ascertain the correctness of a system design - if it is behaving properly, in a correct fashion, and if it is functioning according to a set of requirements; *further analysis* of a formally specified system can provide significant insights into its internals such as its performance; and with proper and mature tools available, the formal specification can be used directly or indirectly as a guide to the final *implementation* of the system. However, current formal methods do not provide features suitable for representing a comprehensive view of multi-agent systems. Either they are geared towards representing its evolving configuration and structure, or they are used primarily to reason about the behavior of the agents. Both these aspects are crucial when dealing with multi-agent systems, and cannot be effectively studied in isolation of the other.

In this paper we present a formal modeling language called CAML that will fill this void. CAML is a process algebra that is inspired partly from -calculus based formalisms and also the popular Belief-Desire-Intention theory of modeling the cognitive skills of agents to represent their behavior. The result is a formal framework that provides the proper primitives and constructs to model and analyze a typical multi-agent system.

2. CURRENT STATE OF THE ART

The last couple of decades have seen a growing interest in the development of formal foundations for multi-agent systems. Due to the complexity, non-determinism, and variety of applications, no single theory or framework has claimed prominence, with the possible exception of BDI. Not only that, a number of "formal" tools cannot really be termed as such, in the stricter meaning of the term. At least the following conditions should hold: the formalism should have a system specification language with *multi-agent system* related *constructs* for abstractions such as agents, agent communication, reasoning, system hierarchy, and

1-4244-0945-4/07/$25.00 ©2007 IEEE

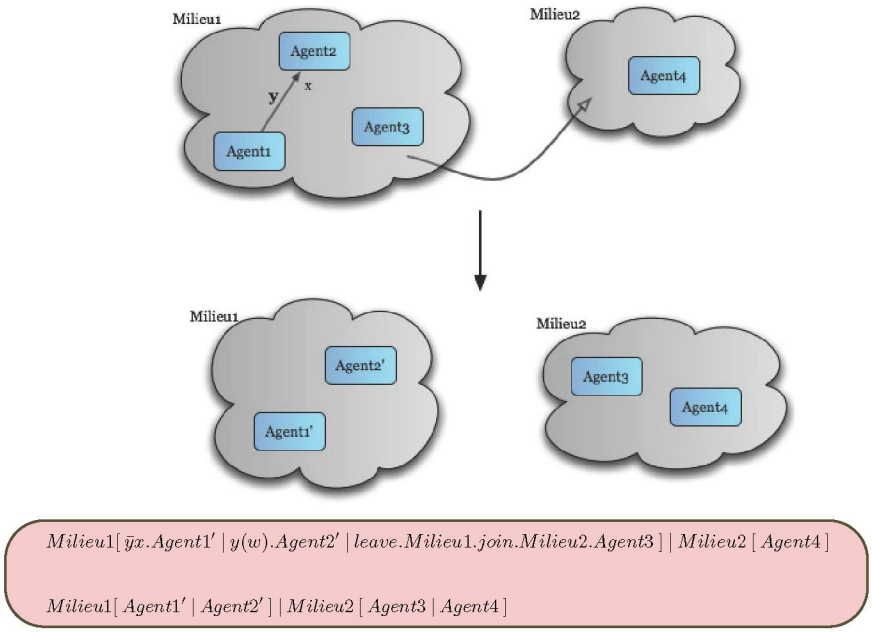


Figure 1. Structural representation of a multi-agent system and its expression in API-Calculus

agent mobility; it should have a complete set of *operational semantics*; and these semantics should allow for *verification* either through system refinement or automatic model checking. Several formalisms that are currently in use in this field do not meet these requirements. These can be classified as: specification languages originally meant for software engineering, such as VDM [7], ETL [3], and tools and languages built around diagrammatic system specification, including UML-related tools such as AML [4], and PASSI [5].

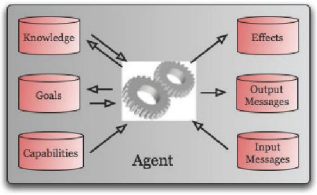
The major approaches towards formal modeling and analysis can be divided into two broad classes in terms of the particular aspects of multi-agent system they are geared towards:

a) Formalisms for modeling structural organization: multi-agent system tend to have an incredibly complex structure because of a number of reasons: the number of agents, the creation and destruction of agents, the evolving nature of the communication structure, possible mobility of agents, and issues of coordination and cooperation between agents, among other complexities. Traditionally, system structure has been studied best using formalisms which support concurrent computation such as Petri Nets, Actor model, and process calculi. Process calculi that extend pi-calculus [10], itself an extension of CCS [11], have been particularly popular for describing systems with concurrent computing elements. These calculi treat the communication of *names* as the basic primitive and pi-calculus itself is known to be Turing complete [12]. Although this class of formal methods is suitable in modeling aspects of concurrent computation, to model the reasoning, behavior, and planning, inherent in all multi-agent system, would be difficult if not totally impossible, due to the unavailability of any high-level constructs for this purpose. Figure 1 shows a typical example of how formalisms of this class represent multi-agent systems.

b) Formalisms for modeling behavioral aspects: Possibly the more exciting part of multi-agent system research is the analysis of agent behavior, reasoning, and planning. Indeed, this is where the intelligence of individual agents come into play - decades of artificial intelligence research provide a solid foundation here. The formal methods that have gained prominence in this area have their foundation in logic, specifically modal logic such as temporal and epistemic, that model agent reasoning as a set of its cognitive skills. This includes the seminal work done by Rao and Georgoff [13] on formalisms for the Belief-Desire-Intention (BDI) model. BDI has since then formed the core of several research undertakings for reasoning about agent behavior in both formal and practical contexts [14]. Figure 2 shows a typical representation of a multi-agent system by formalisms of this class.

What's Missing

Both classes of formalisms discussed above suffer from several shortcomings that make it hard to adopt any of the formal methods available today as a comprehensive multi-agent system modeling tool. The two classes represent a significant divide in objectives that has not been bridged. The structural organization and behavior of a multi-agent system are both integral to the understanding and analysis as well as the implementation process. Class a) has a significant drawback in itself: although pi-calculus and its extensions have enjoyed considerable success for specifying and reasoning about concurrent systems, the complexity of multi-agent systems requires a much more sophisticated framework than what they offer, including high-level abstractions such as those for agents and communication. There have been some efforts in this




```
defineAgentClass Client(?prod,?s) {
  knowledge = {money(?s)}
  goals = null;
  capabilities {
   searchProduct {
capability for searching the departments selling a certain product
     message=search(?pr);
     condition=null;
     do{send(?Agl:EMarket,needProduct(?pr)) }
     effects=null;
    }
    foundProduct {
capability for adding to the knowledge base the names of the interesting domains
     message=haveProduct(?pr,?price);
     condition=null;
     do{send(this,tell(sell(sender,?pr,?price)).Java(EM.wait(20)).
      send(this,goShopping(?pr)).send(this,tell(migrating())) }
     effects=null;
    }
    goShopping() {
capability for migrating to all the domains that sell the product and buying it
     message=goShopping(?pr);
     condition=not(hasKnowledge(migrating()));
     do{forAllKnowledges(sell(?d,?pr,?price)){
        move(?d).send(?d,buy(?pr)).
        changeNumKnowledge(money(?m),1,-,?price)}.
       move(authority)}
     effects=null;
   }
  }
  processes={send(this,search(?prod))}
}
```

Figure 2. Behavioral representation of an multi-agent system and its expression in CLAIM

direction, most significantly Ambient Calculus [16] and its extensions, and API-Calculus. For class b), the BDI model of agent behavior and planning has been recognized as a very suitable foundation. However, none of the BDI based formalisms or specification languages deal with the configuration and structure of multi-agent systems.

Two formal methods have specific relevance to our work: the first is psi-calculus [9], a process algebra that intends to formalize, in an abstract manner, the plan execution model of agent computation that is common to several BDI-based frameworks such as PRS [6] and dMARS [8]. However, it does not deal with the configuration and composition of the agent system as a whole. Another process calculus that makes a significant attempt to provide a comprehensive formalism is CLAIM [CLAIM]. CLAIM meets the need of tackling both the configuration and reasoning. Specifically, the aspects of hierarchical composition of agents, agent mobility, and agent communication are handled in depth. However, the model of agent reasoning is not mature - agent execution is carried out by message communication, and initiation of agent methods. Although this kind of a model helps in translating a formal model of a system into an executable implementation, it does away with the more realistic and adaptable reasoning models of BDI and its extensions.

In our opinion, the unavailability of a single formal method which can be used for both the structural and behavioral analysis is a crucial aspect that needs to be addressed in multi-agent system research. At least part of the problem is the complexity of such an undertaking. Besides the task of composing the right syntax for such a formalism (which should deal with the sticky problems of what an agent and a multi-agent system is), the major work will be in working out the operational semantics. The latter can be extremely complex if the operational semantics of the much simpler calculi such as API-Calculus and Ambient Calculus are any indication. Also, any formal method that hopes to be adapted in the field of multi-agent system, should provide a concrete methodology for verification of the systems that can be described using it. These may include basic techniques such as bisimulation or more advanced ones like model-checking.

3. A COMPREHENSIVE INTELLIGENT AGENT MODELING CALCULUS

In Fig. 3, a comprehensive view of a multi-agent system is presented. It is obvious that such a view will be closer to the reality of implemented systems than the ones in last section - it recognizes the necessity of representing not only an agent's behavior (its intelligence/reasoning) but also how that agent is situated in its environment in terms of its hierarchy, mobility and communication. Although the choice of primitives can always be debated upon, the intention of developing a formal language that is able to allow the representation of all relevant and characteristic features of multi-agent system has not been shared by other formalisms in the past. Of course, the involvement of such a variety of constructs makes the calculus much more complicated than the ones we have described in the last section, but we believe that the availability of the calculus will be of tremendous benefit in the near future, specially as a foundation for more high-level tools that are easier to interact with. This includes the development of a high-level language that uses the operational semantics of the formal language and provides a much more usable syntax and programming ability, and also a visual development environment that will allow defining and composing CAML based agents. Also, the availability of a robust and comprehensive formal language like CAML will serve to foster a more constructive dialog between researchers who are involved in the different aspects of multi-agent system with a more holistic outlook. In practice, it will also advance the development of more robust and easily analyzable multi-agent system-based applications.

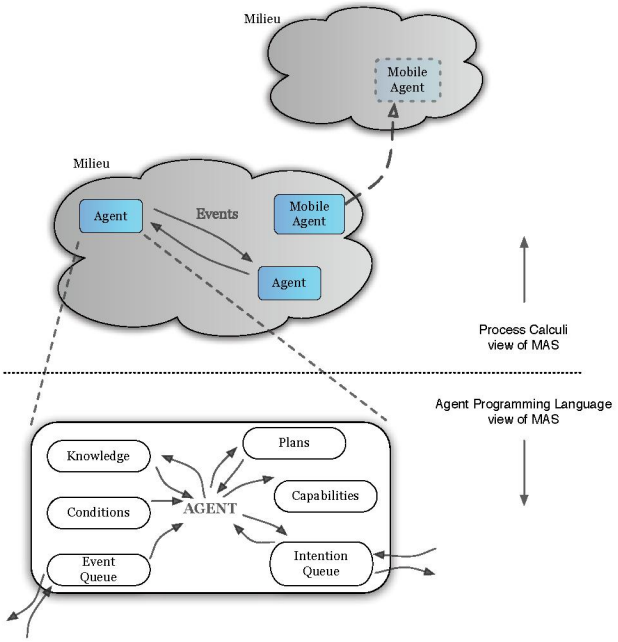


Figure 3. A comprehensive view of representing Multi-Agent Systems

The proposed formalism, named Complete Agent Modeling Language (CAML), is a process algebra that promises a formal framework in which a comprehensive model of multi-agent systems may be represented, including its organization and the reasoning ability of its constituent agents. The syntax and semantics of CAML are not based on any other formalisms (which is true for some other languages such as API-calculus, -calculus or SEAL). With the tradeoff of more work in the development of semantics, it allowed for much more freedom and flexibility. For modeling the organizational aspects of a system, CAML provides single-level grouping primitives named *milieus*. A milieu can constrict agents in a boundary which defines a specific execution entity as well as restricts the visibility of an agent to those in its parent milieu. For modeling the execution and reasoning abilities the relevant primitives are *knowledge*, *events*, *actions*, *conditions*, *event queues, intention queues, and actions* - constructs that are inspired by the BDI model of cognitive skills and plan execution, also adopted in agent-oriented programming languages such as AgentSpeak [15]. For the sake of brevity, we only present the syntax and an informal semantics of CAML in this paper; a more detailed treatment of the syntax and a formal operational semantics based on reduction rules such as those defined for API-Calculus will follow in a future paper.

4. SYNTAX AND SEMANTICS

In this section, we present the syntax of CAML and an informal semantics that will give the reader an understanding of the mechanics of the language.

As detailed later, an agent is composed of its *knowledge*, *Conditions*, *Capabilities*, *Event Queue*, *Plan*, and *Intention Queue*. These basic primitives make up the reactive and proactive behavior of an agent.

Knowledge, K. The knowledge construct is a repository for information that reflects an agents view or belief of its environment including other agents. K is an abstract representation of the agent's belief state and does not provide the composition of individual belief items. The change in agent's knowledge in the progress of a multi-agent system's execution is handled by knowledge transitions. These transitions form the core of defining the agent behavior and are described in detail later.

Events, E. An event can be external (initiated by another agent message, change in environment) or internal (change in an agent's knowledge). These events initiate the execution chain of agents by triggering actions under the guard of contexts. Again, an event is an abstracted entity and does not represent any internal composition.

Conditions, C. A condition acts as a guard for triggering an action for an agent. Each condition acts as a post-condition that needs to be satisfied in order for an action or actions to take place for the occurrence of an event. Every agent keeps a set of conditions which hold true in a particular state.

Actions, Act. An action represents the tasks an agent can perform in the future. Actions are modeled as processes in the tradition of process calculi based on -calculus. An action in CAML can be:

- τ, an internal action
- τ_{κ}, internal knowledge transition
- *join(m)*, agent joins milieu *m*
- *leave*, agent leaves its parent milieu
- *send(e)*, agent sends event *e* as a broadcast

τ, the internal action represents any action that is not explicitly defined such as those required for internal housekeeping. τ_{κ} is an explicit knowledge transition of the agent, instead of those initiated by the an external event. The *join(m)* and *leave* actions represent the mobility of the agent and instruct the agent to join milieu *m* or leave its parent milieu. The action *send(e)* broadcasts the event e to its environment and replaces the low-level send and receive actions of -calculus based algebras. The action is not directed towards a particular agent in the environment, rather it is supposed to represent an event that takes place in multi-agent system to which any agent can respond depending on its own capabilities. This also makes a receive action unnecessary. The significance of these actions will be apparent in the next sub-section when we describe the semantics. Actions can be composed by the the following operators: . , | and + which result in sequential, parallel and non-deterministic execution.

Knowledge Transition, . Knowledge or an agent's belief system plays a crucial role in its behavior. The transitions between the knowledge states represents the changing

beliefs of the agent either due to internal actions and events or external events. A transition is written as:

$$k_1 \xrightarrow{e|c|a} k_2$$

where the transition from knowledge state k_1 to k_2 is qualified by the occurrence of event *e*, under the condition *c*, and results in the action *a*. Either of the three qualifiers can be null or compound constructs. For example, a transition with a null action implies that the event is only meant to bring about the transition in knowledge of the agent and no actual execution needs to carried out; similarly, a transition with a null condition implies one without any restriction. Although, it is named as such, the transition does not have to actually result in a change in the agent's knowledge, i.e., k_1 can be equal to k_2 in which case the purpose for the event is the action alone.

Capabilities, Q. The capabilities of an agent represent the possible transitions that can occur in that agent. Therefore the capabilities is a set of transitions .

Event Queue, Eq. Any event that needs to be handled by an agent is acknowledged by adding it to the event queue.

Intention Queue, Iq. This serves as a temporally task buffer. Any action that an agent intends to take is added to its intention queue.

Plan, P. A plan is what drives an agent's execution according to what it needs to achieve in order to satisfy its goal. In this sense and how it is implemented in the semantics, it can be seen as a cross between *plan* and *desire* of the classical BDI model. Specifically it is an eventless transition $k_1 \xrightarrow{c|a} k_2$ where the transition takes place by the satisfaction of the condition alone.

Agent, A. An agent is a composition of some of the constructs defined above. Specifically, an agent is defined as:

$$A \equiv [K, Q, C, P, EQ, IQ]$$

Therefore, an agent can be identified by specifying its knowledge, its capabilities, the conditions that hold true, the set of plans that will let the agent achieve its goals, and its intention and event queues. An agent changes its states throughout its lifetime as a result of the transitions of its constituents. Therefore, the state space of the four constructs reflects the agent configuration in CAML.

Milieu, M. A milieu acts as a grouping container that also constricts its agents in terms of execution and communication. Inspired from API-Calculus [18], it will provide an isolated computational unit where agents can reside and execute. Agents can join and leave a milieu, and the communication with agents external to the milieu is restricted. Milieus provide the basic support for mobility as well, and will be able to model different real-world entities such as a single computer, a network, an execution-sandbox, etc. It can be both, an abstract idea (a milieu of agents that are responsible for a very specific task), as well as a physical reality (agents present in the local network). It is devoid of any capabilities to perform actions, and other agent-specific tendencies and in that sense it is also different from the idea of an environment, which is supposed to act as more than a container like a milieu does.

An Informal Semantics of CAML

A complete and formal operational semantics is not provided in this paper for the sake of brevity and clarity. It includes basic reduction rules in the style of -calculus that describe the execution of agents as guided by events and plans, and also communication and configuration aspects of the multi-agent system such as mobility across milieus. In the following, we describe a more informal semantics of the language with the aim of giving the reader an idea of the intention behind the choice of the primitives, how the composition of these primitives constructs a multi-agent system, and the mechanics and dynamics of agent execution and configuration.

Agent Execution. The core of agent execution revolves around the knowledge transition. As mentioned earlier, the transition $k_1 \xrightarrow{e|c|a} k_2$ is the result of an external event *e* which in the presence of condition *c*, will lead to the knowledge transition of k_1 to k_2 itself and also the action *a*. An external event, when it occurs, is added to the agent's event queue. The selection of an event from the queue depends on the *selection policy* of the particular agent, and can be non-deterministic, ordered temporally or by a priority value assigned to the events either externally or internally. After the selection of an event, it is matched syntactically with the agent's capabilities. When a match is found with a particular transition's triggering event, the transition is then said to be *initialized*. At this point, the condition *c* is matched with the conditions in C and if the condition holds true, then the knowledge transition takes place and the action *a* is added to the intention queue. The intention queue works under the same selection principle as the event queue, and tasks (actions) are selected for execution.

The event based execution of tasks described above is solely for external events triggered by other agents or by the change in the system configuration. Tasks triggered internally are guided by the agent's plans. At every iteration of system progress, each plan of an agent is checked for possible execution: if the eventless-transition's condition is satisfied, the knowledge transition takes place along with the addition of the corresponding action to the intention queue. As for the actions triggered by external events, actions in the event queue corresponding to plan execution can include the *send(event)* action, which is responsible for inter-agent communication.

Agent mobility. Agent mobility is handled by the two action primitives, *join* and *leave*, which of course can be activated

by either an event or plan execution. This allows an agent to move around milieus and gives it different contexts for execution.

5. Conclusion and Future Work

We have presented a new formal language for specification of multi-agent systems. The stress is on providing a comprehensive and holistic outlook towards viewing agents - their composition, execution, and mobility. The syntax provides provides all the basic primitives for the general representation of a typical multi-agent system. The BDI-inspired behavioral representation is able to model both the proactive and reactive nature of an autonomous, intelligent agent, while mobility and organization is handled by composing agents in milieus.

We intend to present a complete operational semantics in an extended paper in the future. A major motivation behind developing CAML has been to eventually provide a mechanism for verification of multi-agent systems. This would require a robust operational semantics, including type-checking that will enforce consistency in system composition, and equivalence relations of both structural congruence and bisimulation varieties. In the long run, CAML is intended to form a foundation for a high-level programming language and also a visual development environment that will allow the design, analysis and implementation of multi-agent systems in a comprehensive and easy to use framework.

References

[1] J. Bowen and M. Hinchey. "Ten Commandments of Formal Methods," Oxford University Computing Laboratory Technical Monograph, 1994

[2] J. P. Bowen and M.G. Hinchey, "Seven More Myths of Formal Methods", IEEE Software, Vol.12, n.4, July 1995

[3] S Conrad, G Saake, C Tuerker, "Towards an Agent-Oriented Framework for Specification of Information Systems," Lecture Notes In Computer Science, 1999

[4] M. Cossentino, S. Gaglio, L. Sabatucci, V. Seidita, "The PASSI and Agile PASSI multi-agent system Meta-models Compared with a Unifying Proposal," CEEmulti-agent system, 2005

[5] R. Cervenka, I. Trencansky, and Calisti. "Modeling Social Aspects of Multiagent Systems: The AML Approach," In J.P. Muller and F. Zambonelli, editors, The Fourth International Joint Conference on Autonomous Agents & Multi Agent Systems (AAmulti-agent system 05). Workshop 7: Agent-Oriented Software Engineering (AOSE), pages 85--96, Universiteit Utrecht, The Netherlands, 2005

[6] M. P. Georgeff, F. Ingrand, "Decision-making in an embedded reasoning system," In Proceedings of IJCAI-89, pp. 972–978, Detroit, MI, 1989

[7] C.B. Jones, "Systematic Software Development using VDM," Prentice-Hall, Englewood Cliffs, New Jersey, 1986

[8] D. Kinny, "The Distributed Multi-Agent Reasoning System Architecture and Language Specification," Australian Artificial Intelligence Institute, Melbourne, Australia, 1993

[9] David Kinny, "Algebraic specification of agent computation," Appl. Algebra Eng. Commun. Comput. 16 (2-3): 77-111, 2005

[10] R. Milner, "The polyadic Pi-calculus: a tutorial," Technical Report ECSLFCS -91-180, Computer Science Department, University of Edinburgh, UK, October 1991

[11] R. Milner, "A Calculus of Communicating Systems," Lecture Notes in Computer Science, Volume 92, Springer-Verlag, 1980

[12] R. Milner, "Functions as Processes," Mathematical Structures in Computer Science, Vol. 2, pp. 119-141, 1992

[13] A. Rao, M. Georgeff, "BDI Agents from Theory to Practice," Technical Note 56, AAII, April 1995

[14] A. S. Rao and M. P. Georgeff, "Formal models and decision procedures for multiagent systems," Technical Note 61, Australian AI Institute, Level 6, 171 La Trobe Street, Melbourne, Australia, June 1995

[15] A. S. Rao, "AgentSpeak(L): BDI agents speak out in a logical computable language" In W. Van de Velde and J. W. Perram, editors, Agents Breaking Away: Proceedings of the 7th 24 European Workshop on Modelling Autonomous Agents in a Multi-Agent World, (LNAI Volume 1038), 42-55. Springer-Verlag, 1996

[16] I. Scagnetto and M. Miculan, "Ambient calculus and its logic in the calculus of inductive constructions," In Proc. of LFM, ENTCS 70.2. Elsevier, 2002

[17] M. Wooldridge, and N. Jennings, "Intelligent agents: Theory and practice," The Knowledge Engineering Review , 10, 115–152, 1995