EXTENDING THE BDI-ASDP METHODOLOGY FOR REAL-TIME

Susel Fernández Melián
Department of Automatic, University of Alcalá
Edificio Politécnico, Ctra N-II Km. 31,600
28871 Alcalá de Henares, Madrid, SPAIN
susel@aut.uah.es

Iván Marsá Maestre
Department of Automatic, University of Alcalá
Edificio Politécnico, Ctra N-II Km. 31,600
28871 Alcalá de Henares, Madrid, SPAIN
ivmarsa@aut.uah.es

Ukrania Díaz Rosario
Department of Automatic, University of Alcalá
Edificio Politécnico, Ctra N-II Km. 31,600
28871 Alcalá de Henares, Madrid, SPAIN
ukrania@aut.uah.es

Miguel A. López-Carmona
Department of Automatic, University of Alcalá
Edificio Politécnico, Ctra N-II Km. 31,600
28871 Alcalá de Henares, Madrid, SPAIN
miguellop@aut.uah.es

ABSTRACT

Multi-agent systems are an emerging research area which is experiencing a fast growth. In the last years, many theories, architectures, languages and platforms for the development of agent based systems have been developed. On the other hand, real-time systems represent an important challenge from the perspective of multi-agent systems, considering the increasing need to count on software which is able to respond to certain situations in a timely partition. Nevertheless, in spite of its increasing interest, an important difficulty when applying these technologies to the resolution of a concrete problem is in the agent-based software development process. Many efforts have been made to extend the capacities of standard software modelling to integrate multi-agent and real-time systems, and different methodological proposals exist, but their practical application is not always obvious from the definition of the methodology. This work proposes an extension of the BDI-ASDP methodology for the inclusion of timing constraints. We have applied this methodology to model some of the agents which participate in a virtual baseball game.

KEYWORDS

Real-time Systems, Multi-agent Systems, Software Engineering, Modeling.

1. INTRODUCTION

Agent technology arises as a natural extension of component based solutions, and it has the potential to cause a great impact in the information society, from information processing to automatization of tasks to release the users from the routine tasks they should perform otherwise [13]. There are different application domains where agent technology may play an important role: service personalization in smart environments [21], electronic commerce [3], or the management and control of communication networks [7], [12]. In spite of the

multi-agent systems community research effort, the agent technology has not been widely accepted in the industrial scope. This is partially due to the absence of established and consolidated methodologies for the development of multi-agent systems [14]. These methodologies should include all the phases of the software life cycle, specially the analysis and design phases, since these are the stages where the use of the agent paradigm implies a greater difference, and where the modelling of the problem is more difficult: to decompose the problem in small pieces, to map the different functional or semantic units into different agents, and to define the behaviour of such agents. Systems design with timing constraints has critical requirements in terms of timing behaviour, performance and security. So, it is very important to have tools which include all the phases of the software development (from the specification, analysis and design, to the code generation). Several methodological proposals for software engineering exist which can be applied to agent based systems. Some of them arise from the field of knowledge based systems [8], others directly focus in the properties of the agents [17], and others are extensions to Object Oriented software development methodologies and languages, like UML [16]. Only a few of these methodologies include the temporal behaviour. They do not count either on the support of specific CASE tools, which makes the application of a methodology complex, especially in the initial phases of the modelling processes. Sometimes, to have a practical example or a case of study of these methodologies turns out very useful to solve a specific problem. In this work we choose an agent based software methodology and add the capability of modelling timing constraints, through timing diagrams in the design phase. We have applied this methodology to model some of the agents who would take part in a virtual baseball game.

The rest of this paper is organized as follows. Section 2 introduces the different agent based architectures, and it focuses in the study of the BDI architecture. Section 3 covers the problem of agents in real-time systems, and Section 4 describes the proposed methodology and diagrams. Finally, Section 5 presents the practical application of these tools to our case of study: the baseball game and the last section summarize the work and the main conclusions.

2. AGENT ARCHITECTURES

Interaction and cooperation among agents is a central point in the agent paradigm. However, the starting point of the design and modelling tasks is the agent itself. Some of the most difficult questions to solve when developing an agent based system are: which class of software entity our agents are, which agents are needed, and which is their internal architecture [14]. From the modelling point of view, the importance of the agent architectures is double: firstly, they provide the opportunity to explain and predict the agents' behaviour from the current state of the agents and the state of the environment. Secondly, they allow the development of methodologies to build real-time agent based systems. Typically, we can classify agent based architectures in three different categories. The reactive architectures are based on stimulus-response rules, and they do not provide a symbolic representation of the environment. The deliberative architectures define a symbolic model of the environment, and act in accordance to this model. Finally, hybrid architectures let agents act in a reactive or deliberative way.

2.1 Deliberative Architectures. The BDI Architecture

Agents which are able to maintain and to manipulate representations of the world that surrounds them, and whose behaviour does not obey to stimulus-response rules are called deliberative agents. The behaviour of these agents is characterized and described by means of mental attitudes, such as suppositions, motivations or plans. For many reasons, to model an agent using mental attitudes is very useful. Firstly, the mental attitudes are familiar concepts to the human beings, which makes simple to explain and predict the behaviour of agents. On the other hand, the understanding of the relation between the different attitudes and how they affect the agent may provide mechanisms to define intelligent behaviours. Finally, agents designed in this way could interpret the other's behaviour with independence of the implementation. A Belief, Desire, and Intention model (BDI) does not represent a specific system, but it provides the basis of other several architectures of agents, that's why it can be considered an abstract architecture. The most used agent based architectures are described in terms of the BDI model, in which agents continuously monitorize the environment and act in order to change it. Actions are a consequence of agent's beliefs, desires and

intentions, which represent, respectively, information, motivation and decision making capabilities [14]. The architectures based on the BDI model represent the beliefs, desires and intentions of the agent like data structures that determine the behaviour of the agent. The basis of the model is that an agent has beliefs on the world and desires to satisfy, which takes him to formulate intentions to act. Several formal models for the analysis of the relation between beliefs, desires and intentions have been described [10], [22]. Also, different proposals of architectures and modelling languages exist for the agent design using the BDI model [23], [6].

3. REAL-TIME AGENTS

In the last years, the use of the multi-agent systems paradigm in real-time scenarios arises from the capabilities required for the new real-time systems. This paradigm tries to incorporate flexibility and distribution in a real-time framework. A Real-Time System (RTS) is a system in which the correctness depends not only on the logical result of computation, but also on the time at which the results are produced [18]. According to these concepts, a real-time agent is an agent with timing constraints. These constraints can be hard, soft or both. A real-time agent must guarantee the fulfilment of timing constraints, and must try to reach its goals concurrently. Real-time artificial intelligence systems (RTAIS) have emerged like useful techniques to solve complex problems that require intelligence and reaction in real time. In new real-time systems the flexibility, adaptability and intelligence properties of behaviours are some of the most important ones. Thus, the agent paradigm seems to be appropriate to develop real-time systems in hard real-time environments.

4. THE METHODOLOGY

The primary goal of this work is the extension of a software development methodology based on BDI agents, in order to allow the modelling of real-time multi-agent systems, and its application to a practical example. A software development methodology generally covers two main different aspects. Firstly, a methodology uses a modelling language which is used to describe the models and to define the different elements in the models with a standard and coherent notation. A methodology is defined by means of a software development process, which specifies the activities and their relations during the software life cycle, as well as the products of each activity. In our study case, we have used the BDI-ASDP development process (BDI- Agent Software Development Process) [4], which is centred on the decomposition of the problem in beliefs, desires and intentions, and we have added the timing diagrams proposed in UML 2.0 [19] to model real-time systems.

4.1 UML and the Modelling of Agent Based Systems

UML (Unified Modelling Language) is one of the most used tools in the area of software development. UML provides developers with the mechanisms to capture ideas in a convenient way. The communication of these ideas is a matter of great importance in software development, and UML provides the capability of organize the design process in an understandable and unified way for analysts, clients, developers, and everyone involved in the software development. The most recent UML version (UML 2.0) does not provide the structures to express concepts like goals, agents, groups, multicasting or generation functions which are used in agent modelling. Thus, researchers and developers have not agreed upon a unified form to represent agents and multi-agent systems. So, it is highly probable that the appearance of a unified agent-based modelling language with the richness and acceptance of UML will be delayed several years [1]. Nevertheless, the FIPA Modelling Technical Committee, together with the OMG Agent Interest Group, is carrying out noteworthy efforts for extending UML 2.0 in order to model agent-based systems.

4.2 UML and Real-Time Systems

The design of real-time systems has critical requirements in terms of timing behaviour, performance and security. It is fundamental to have tools that include all the phases of the development (from specification, analysis and design, to the generation of application code). UML 2.0 incorporates several models to make easy the modelling of systems with timing constraints, through new diagrams that help to obtain a better representation of software with these characteristics. The standard UML was poor in the documentation of real-time applications and concurrent systems, until the real-time profile was adopted. Now, UML supports primitive constructions such as "resources" which are modelled altogether with the OO standard. There are a number of suitable techniques within the UML specification to specify and design real-time systems. In order to model real-time systems in UML, the state and activity diagrams have been used, and the timing diagrams have been added now. The intention of UML in real-time systems is to provide a minimum, but sufficient, number of extensions to standard UML to support all the techniques identified in the perspective of real-time systems, in order to promote the standardization at the industry level of the proposed extensions. The OO modelling appears like a powerful tool to capture the different characteristics of systems with timing requirements. In this context, UML is widely being used. The UML extension [19] is used to provide visual modelling techniques for the development of real-time systems. These extension mechanisms are allowed within UML like "presentation options". UML allows that different icons are used for the same element of the model.

4.3 The Use Case Based Development Process of BDI Agents

The key point of the used methodology is the use of the BDI architecture for the description of the agents: beliefs, desires and intentions of each agent are assigned. The process provides the necessary tools to make the extraction of mental attitudes for the agents in a systematic form. At the beginning of the development, external use cases are used, so that they constitute general schemes of how certain services can be provided from an external point of view. Later, by means of the use of internal use cases, these schemes are described in detail and each service is broken down into one or more objectives. The internal use cases also provide a detailed description of each objective and its corresponding planning (intentions). Once define the objectives and their associated intentions, are defined the necessary beliefs extracted so that the defined objectives may be reached. These beliefs arise from the use of Data Flow Diagrams (DFD) applied to the planning of each objective. Thus we can observe the movement of the data and determine which beliefs are necessary to complete each plan. BDI structures created in this way are assigned to different agents. Figure 1 describes the BDI-ASDP development process in detail [4], specifying in addition the tools that are used in each stage of the process. The development begins with an initial statement of the problem, which reflects what the system must do. Later, a brief description of external use cases is used, where the services that the system must provide are defined. In a detailed description of external use cases those services are described as goals, and a planning is made from an external point of view. Analyzing these use cases a goal hierarchical diagram can be made. At this point, the analysis phase ends, and the design phase starts. The next step is to define internal use cases, which concerns interactions among elements inside the system. They can show how entities interact to the system internally and how the entities use each other to get things done. The scenario of each service is decomposed into details that help us find plans for the goal. For each use case, the sequence diagrams are defined to illustrate the order of occurrence of the events and the communication between the objects by means of messages. Later, these sequence diagrams are translated into activity diagrams, which show the steps or activities that happen during an operation or process. Thus the capturing planes stage in the design phase concludes, and the phase where the beliefs are obtained begins, where a list of beliefs is created identifying the beliefs that are necessary so that the plans associated to the objectives can be completed. The final phase of the process is centred in describing the agents in such a way that makes its software construction simple. To achieve this, it is necessary to map the blocks of beliefs, desires and intentions to agents. The obtained interaction diagrams in the previous phases help to make this mapping. Also, BDI cards for each agent are created, so that two representations of the system are obtained, related respectively to the dynamic structure of the agents of the system (its interaction) and to its static structure (the set of beliefs, desires and intentions).

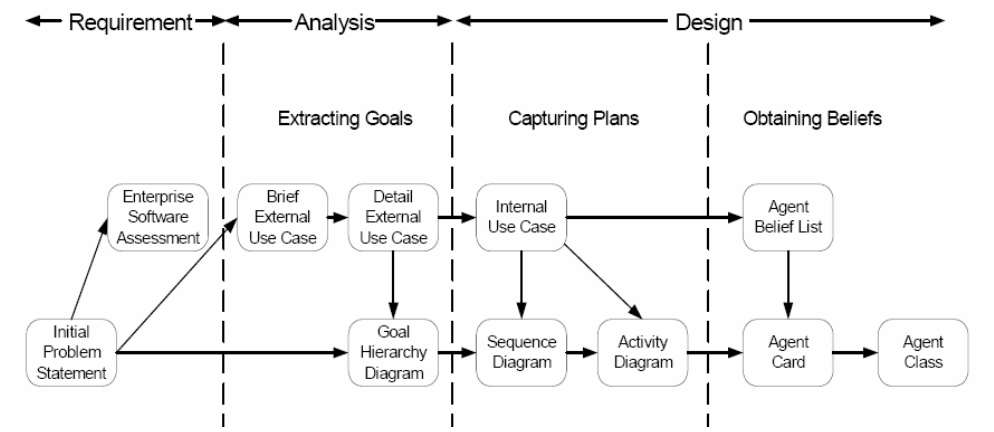

Figure 1. BDI Agent Software Development Process

4.4 Timing Diagrams in BDI-ASDP.

The BDI-ASDP agent based software development process proposes several stages for the discovery of beliefs, desires and intentions of the agents which compose the system. In each stage the corresponding diagrams are developed in order to model the behaviour of the agents. Considering that ASDP does not cover the possibility of modelling agents with timing constraints (i.e. real-time agents), we propose to include UML 2.0 timing diagrams in the capture of plans stage of the ASDP design phase. In this stage the internal use cases are defined, and the sequence of actions that take place are detailed through the activity and sequence diagrams. However, these diagrams are not valid to model the timing behaviour of agents, since they focus on the logical sequence of the activities and not in their duration, which is the most important issue when we have to consider agents which have to execute actions under time constraints. So, timing diagrams are used to explore the behaviour of one or more objects through a period of time. These are a form of interaction diagram that is centred in timing constraints which can be shown in two alternative forms, both representing the same kind of information [20]. As Figure 2 shows, the main difference between them is that state transitions are described by means of vertical movements in the first alternative (Figure 2.a), where each horizontal line is a state, whereas the second alternative represents state transitions with a cross, where each state is the area between two parallel horizontal lines (Figure 2.b). Timing diagrams are useful to show time restrictions between state transitions of different objects. These are often used in the design of embedded systems [5].

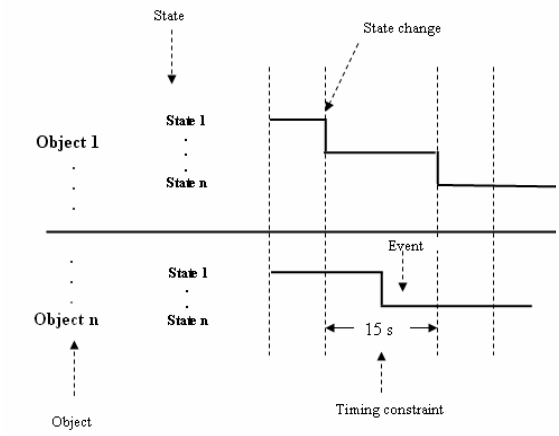

Figure 2.a) Timing diagram showing states as lines.

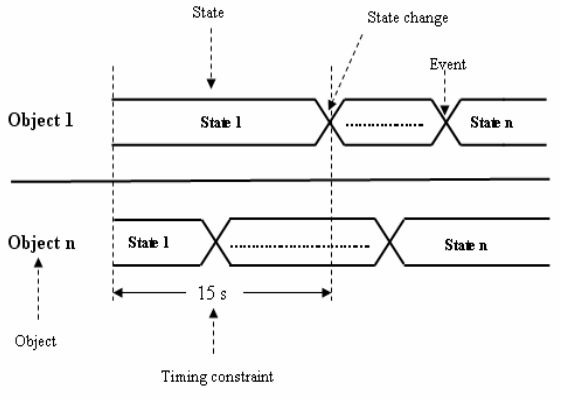

Figure 2.b) Timing diagram showing states as areas.

5. APPLICATION SCENARIO

5.1 The Baseball Game

Baseball is a game which is played with a hard ball and a bat, where two teams and nine players per team participate. Baseball is played in a diamond shape field. One of its corners it is marked by a rubber piece, called home plate. In the other three corners of the field, moving from the home plate in counter-clockwise are the first, second and third bases, each one marked with a pad. A baseball match is divided into nine periods which are called innings. The team which scores more runs throughout the nine innings wins the match. The game begins when a pitcher throws the ball towards the catcher. The batter of the opponent tries to bat (to strike with the bat) the ball towards the inner field. Players of a team score runs batting the ball and running around a series of bases until one of these players is eliminated by the other team.

5.2 The BDI-ASDP Model in the Baseball Game.

This section shows the practical application of the BDI-ASDP process to our case of study. For reasons of space, the complete model of the system is not presented, but only the diagrams that allow to model timing constraints at one of the main use cases of the system: the use case that represents the pitch action.

5.2.1 The Pitch Action.

The pitch action is one of the most important actions which take place in a baseball game. In this action all the players are somehow involved, but fundamentally the pitcher, the catcher, and the batter. Initially, the pitcher and the batter are in the preparation stage. Soon, when the pitcher is ready, he throws the ball towards the catcher, and the batter tries to strike the ball and run towards the first base, whereas the pitcher assumes a defensive position. All of these actions must be made in a synchronous way in order to be able to achieve the success in the game. The pitcher has preparation time in which he must agree with the catcher about the type of throwing that is going to carry out, whereas the batter must be ready to move the bat at the precise time to strike the ball with effectiveness.

5.2.2 Timing Diagrams for the Pitch Action:

Timing diagrams are useful to show timing constraints between the changes of state of different objects. For the accomplishment of the pitch action timing diagrams we have defined three different objects: the pitcher, the batter, and the ball, and their corresponding states that they will run through. Figure 3 shows the timing diagrams of the pitch action.

The Pitcher States.

- Preparation: This state represents the moment at which the pitcher prepares itself to pitch.
- Ready: This state represents the moment at which the pitcher is ready, awaiting the signal to carry out the pitch.
- Sending: This state represents the moment at which the pitcher is carrying out the pitch.
- DP: It represents the defensive position that the pitcher assumes after the batter strikes the ball.

The Ball States.

- In rest: This state represents the moment at which the ball is in rest.
- In the air: This state represents the moment at which the ball is in the air. It can happen when it has been sent or when it has been struck by the bat.
- Struck: This state represents the moment at which the ball is struck by the bat.

The Batter States

- Preparation: This state represents the moment at which the batter prepares itself to receive the pitch.
- Ready: This state represents the moment at which the batter is ready, awaiting the pitch.
- Swing: This state represents the moment at which the batter is making the movement to strike the ball.
- Running: This state represents the moment at which the batter is running, after having struck the ball.

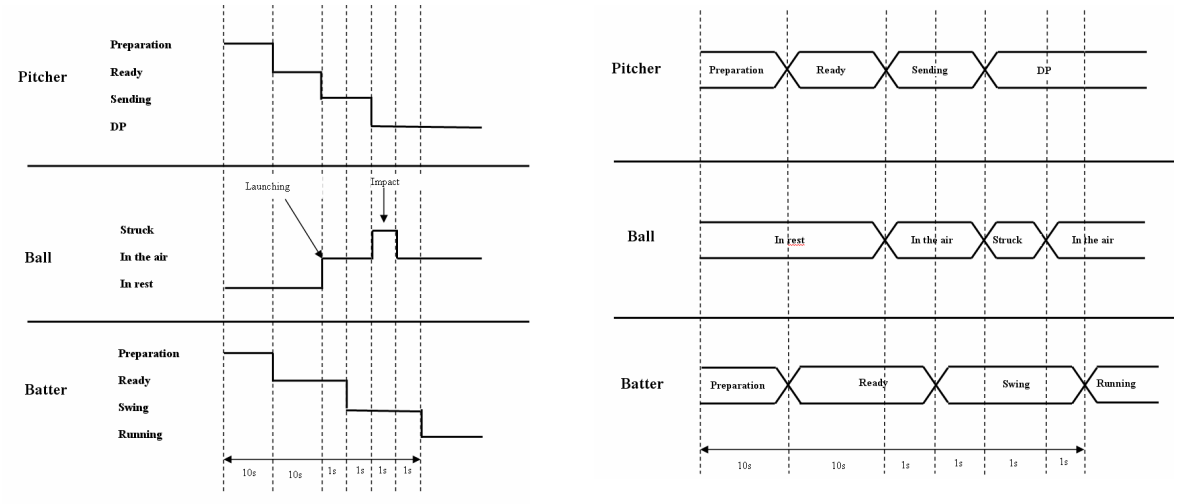

Figura 3. Timing Diagrams of Pitch

6. CONCLUSION

It is clear the importance and utility of modelling timing restrictions in a structured and rigorous form and in agreement with an established methodology. Moreover, it is very convenient to have CASE tools in order to attempt the development processes according to these methodologies. However, at present there are not specific tools and generic modelling tools must be used, with the consequent difficulties of adjustment and adaptation to the particular methodology used. The appearance of modelling tools for agent based software development is highly coupled to the existence of consolidated methodologies which standardize the different phases of agent based software development. There exist many development methodologies in the area of multi-agent systems, but there is no a global agreement yet. However, important efforts are being made as much in the design of methodologies like in the development of tools that support them, and we hope that in a few years, agent based software development will reach the necessary standardization level to extends its deployment at all levels.

ACKNOWLEDGEMENT

This work is part of the IMPROVISA Project. It has been supported by the Education and Science Ministry. MEC-TSI2005-07384-C03-03.

REFERENCES

[1] B. Bauer and J. Odell, March 2005 “Uml 2.0 and agents: how to build agentbased systems with the new uml standard,” *Journal of Engineering Applications of Artificial Intelligence*, vol. 18, no. 2, pp. 141–157.

[2] R. Brooks, 1986, “A robust layered control system for a mobile robot,” *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 14–23.

[3] L. C. DiPippo, V. F. Wolfe, L. Nair, E. Hodys, and O. Uvarov, December 2000, “A real-time multi-agent system architecture for e-commerce applications,” University of Rhode Island, Tech. Rep. TR00-280.

[4] J. M. Einhorn and C.-H. Jo, 2003, “A use case based bdi agent software development process,” *in Proceedings of the 2nd. International Workshop on Agent Oriented Methodologies* - OOPSLA-2003, Anaheim, CA, USA.

[5] Martin Fowler. *Uml Distilled* (3Ed 2004). Addison Wesley.

[6] M. Georgreff and A. Lansky, 1987, "Reactive reasoning and planning," *in Proceedings of the Sixth National Conference on Artificial Intelligence*. MIT Press, pp. 677–682.

[7] V. Gorodetski, I. Kotenko, and O. Karsaev, July 2003, "Multi-agent technologies for computer network security: Attack simulation, intrusión detection and intrusion detection learning*," Internacional Journal of Computer Science and Engineering*, vol. 18, no. 4, pp. 191–200.

[8] C. Ángel Iglesias Fernández, January 1998, "*Definición de una metodología para el desarrollo de sistemas multiagente*," Ph.D. dissertation, Universidad Politécnica de Madrid.

[9] P. Janca, "Pragmatic application of information agents," BIS Strategic Decisions, Tech. Rep., 1995.

[10] N. R. Jennings, 1992, "On being responsible," Decentralized AI 3 – *Proceedings of the Thir European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pp. 93–102.

[11] N. Jennings and M. Wooldridge, January 1996, "Software agents," *IEE Review*, pp. 17–20.

[12] B. Jennings, R. Brennan, R. Gustavsson, R. Feldt, J. Pitt, K. Prouskas, and J. Quantz, 1999, "Fipa-compliant agents for real-time control of intelligent network traffic," *Computer Networks*, 31, pp. 2017–2036.

[13] M. Luck, P. McBurney, and C. Priest, 2003, Agent Technology: Enabling Next Generation Computing. AgentLink.

[14] M. Luck, R. Ashri, and M. d'Inverno, 2004, *Agent-Based Software Development*. Artech House Publishers.

[15] P. Maes, "The agent network architecture (ana)," *SIGART Bulletin*, vol. 2, no. 4, pp. 115–120, 1991.

[16] J. Odell, H. V. D. Parunak, and B. Bauer, 2000, "Extending uml for agents," *in Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, G. Wagner, Y. Lesperance, and E. Yu, Eds., Austin, TX, pp. 3–17.

[17] A. Omicini, 2001, "Soda: Societies and infrastructures in the analysis and design of agent-based systems," *in Agent Oriented Software Engineering*, P. Ciancarini and M. Wooldridge, Eds., vol. 1957. New York: Springer, pp. 185–193.

[18] J. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, 12(10):10{19, 1988.

[19] "Uml resource page", Object Management Group, Tech. Rep, http://www.uml.org.

[20] UML 2.0 *Superstructure Specification*. Revised Final Adopted Specification (ptc/04-10-02). October 8, 2004.

[21] J. R. Velasco, I. M. Maestre, A. Navarro, M. A. López, A. J. Vicente, E. de la Hoz, A. Paricio, and M. Machuca, 2005, "Location-aware services and interfaces in smart homes using multiagent systems", *to appear in Proceedings of the 2005 International Conference on Pervasive Systems and Computing* (PSC'05: June 27-30, 2005, Las Vegas, USA).

[22] M. Wooldridge, "This is myworld: The logic of an agent-oriented testbed for dai," Intelligent Agents: Theories, Architecures and Languages, *Lecture Notes in Artificial Intelligence* 890, pp. 160–178, new York: Springer.

[23] M. Wooldridge and N. Jennings, Eds, 1986, A Pragmatic BDI Architecture, *ser. LNCS. New York: Springer*, vol. 1037, pp. 203–218.