Designing agent-based systems with UML

Massimo Cossentino, Antonio Chella and Umberto Lo Faso,

Abstract - **In this paper a design process for agent oriented software design is presented that uses the Unified Modeling Language. It starts with use-case diagrams in which at a certain level of abstraction some use-case are identified with agents (agents identification phase). Then the structure of the agents is designed in a class diagram. Each class in this diagram contains all the methods that the following description of the agents' behaviors phase proves necessary. These two phases are iterated to introduce all the behaviors. The whole process can be repeated until all the requirements have been implemented.**

I. INTRODUCTION

In the last years there has been a great increase in the number and dimension of agent-based software. Problems that proved difficult to be managed with traditional object oriented solutions have been successfully solved with agents.

We can, unfortunately, observe that instead of such encouraging results in application, the design and developing techniques are slowly growing. Many authors have given very interesting contributes to the argument ([1], [2], [3], [4], [5], [6]) but the discussion is still opened.

In this paper, we look at multi-agent systems and propose a design method that is based upon the well known UML (Unified Modeling Language). We have chosen this language primarily because it is widely accepted both in the academic and industrial worlds and secondarily because it has some extension possibilities (constraints, tagged values and stereotypes) that permits us to better fit the requirements of an agent-oriented design. These extensions proved useful when we had to face some agent specific aspects of the design and helped us to solve some of the problems. Moreover we think that specifically agent-oriented UML extensions are needed and we make a proposal that we compare with the ideas of other authors. In the discussion of our approach we also identify some strategy of use (and therefore of design) for an agent dedicated UML CASE tool.

II. THEORETICAL BACKGROUND

UML is a language that can be used to analyse, specify, construct and document a software artefact. It is not conceived to support a specific design process even though some of the most important contributors to its growth, in the past, have developed well-known methodologies (Booch's OOADA, Jacobson's OOSE, Rumbaugh's OMT and others).

Several different kinds of diagram permits to deal with the different aspects of the object oriented software design.

The use-case diagrams represent the interactions of the entities which are external to the system (called actors) with the system itself which is represented through its functionalities (called use-cases). These diagrams are often used in the analysis phase.

The structure of the system can be depicted through the class diagram in which classes (entities of the system) and their relationships are shown. Classes could contain attributes and methods (addressing the behaviour of the system).

The system's dynamic behaviour can be described using several different diagrams: collaboration, sequence, state and activity diagrams. Collaboration and sequence diagrams give a different point of view of the same scenario. Scenarios are paths around use-cases illustrating one of the possible behaviour of the software. In collaboration diagrams the attention is focused upon the message exchanged between the entities; the diagram well illustrates the communication aspect (who sends something to someone else). Sequence diagrams again show messages but arranged in time order (showing when a message is sent). Traditional finite state and activity diagrams (in which transitions are triggered by the completion of the activities to be performed in each state) can be used to describe a certain procedure or the life of a class.

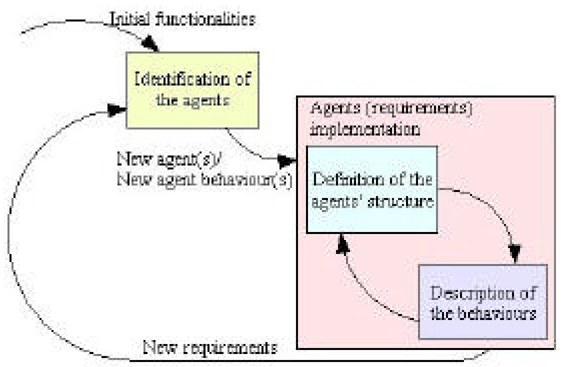
Some coding aspects can be detailed in the component diagrams in which classes are associated with components (executables, libraries, ...) that will be created. At last, in the deployment diagram processes and nodes (execution units or other devices) can be defined together with their connections.

III. DESIGNING AGENT BASED SOFTWARE

Many authors agreed that agent oriented software can be profitably used in order to solve complex problems [17], [18]. In such a context, designers usually face some specific aspects:

a) a complex problem will probably lead to a multi agent solution in which communication (and collaboration) between agents is a strategical issue;
b) it is also possible that different parts of the system will run on different elaboration units;
c) the resulting system is quite complex and a rigorous design method has to be pursued also in order to obtain a flexible, efficient architectural structure (probably hierarchical [6]) .

The authors are with the Dipartimento di Ingegneria Automatica ed Informatica of the University of Palermo, Italy.
E-mail: {maxco, chella, lofaso}@unipa.it


Initial functionalities
Identification of the agents
Agents (requirements) implementation
New agent(s)/
New agent behaviour(s)
Definition of the agents' structure
Description of the behaviours
New requirements
Fig. 1. Iterations in the AODPU (Agent-Oriented Design Process with UML)

We will particular deal with "distributed problem solving system" [8] in which system components are expected to cooperate to achieve the system goal. In "open systems" we should also take into account the arrival of external self-interested agents that could interact with the others; obviously this situation may need a different approach and we will not consider this eventuality.

In drawing a model of a software the designer has the possibility to look at implementation at different levels of details. In [21] Robbins et al. define the fidelity of the model as the distance between the model and its implementation. Low fidelity models result in a problem-oriented description while high fidelity models are more solution-oriented.

In our approach we aim at drawing a model belonging to the second category; starting from the problem description (that gives places to a not detailed, low fidelity model) we will iterate and arrive to a precise high fidelity description of the system implementation.

IV. AN OVERVIEW OF THE AODPU

We can divide the traditional software development process in two fundamental steps: the analysis phase in which system requirements are to be captured and the design phase where the identified system functionalities are implemented. In other words, while in the system analysis phase we focus on "what", in the implementation phase we focus on "how".

Dealing with the design of agent based software, we propose a slight different approach: the AODPU (Agent-Oriented Design Process with UML). We mix the two phases together basing our idea on the fact that the use-case diagram (probably the most significant step of the analysis phase) can properly represent agents and their behaviour (implementation aspects of the design).

Iteration allows successive refinement of the system both from the functional and the implementation point of view. In fact, in several cases, we have realised that a linear development process is not sufficient in order to solve complex problems and that some iterative procedures is needed [7].

In order to deal only with one of the possible iterations we will proceed as follows (see fig.1):

a) **Identification of the agents.** A functional description of the system is provided through an hierarchical series of use-case diagrams. The first diagram (we could consider it as some kind of 'context' diagram) will only represent one use-case (the system), some actors in the environment and any external entity interacting with the system. Other use-case diagrams will give more details on the system. As we will see later, in these diagrams the functions of the system and the agent-based solution will be formalised having in mind the correspondence between use-cases and agents that is demonstrated in the following paragraph [20].

b) **Definition of the agents' structure.** In the use-case diagrams some agents are identified and their roles are described (each agent can play a certain role during his interactions with the others; this role can also be depicted through the involved relationships). At this point of the process, a specification of the structure of each agent can be provided through a class diagram in which the methods of each class correspond to the subtasks that each agent is able to perform. Each agent can play his own role in the system organisation using his own methods and interactions with other agents [8]. In the class diagram the various kinds of interactions can be introduced and typed.

c) **Description of the behaviours.** We can describe the scenarios relative to the use-cases diagrams using some sequence diagrams: by this way we can also detail the agents' behaviour taking into account the time variable that is one of the key factors in real-time problems (for example in robotics). Similarly, we can describe the cooperation between agents in a scenario by the collaboration diagram.
Using activity diagrams we can clearly show the contemporaneous actions of the various agents that cooperate to achieve the system goal. We can describe 'what' each agent is able to do, his behaviour and his interaction with the environment and/or the other agents.

We see the "agent structure identification" and the "behaviour description" as mutually dependent and cyclically performed to define agent implementation.

At the end of this process all the requirements are fixed (for this iteration) and the implementation can start.

A. *Identification of the agents*

The first step in this phase is the identification of the entities external to (and, somehow, interacting with) the system.

From this analysis we will draw a context diagram in which the system will be represented within its own application environment; this description can be helpful when focusing the interactions between the system and the external world.

The following steps detail the previous description further more. We can stop this process when the use-cases depict precise roles each of which can be assigned to a single agent.

We think that the use-case diagram is useful to describe the agent system from a social point of view

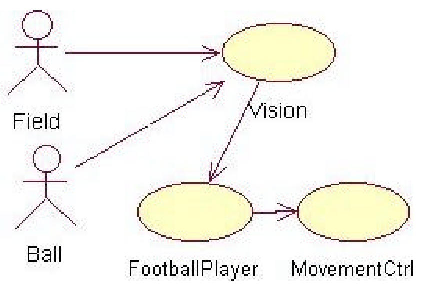


Fig. 2. An use-case diagram of a simple football player robot

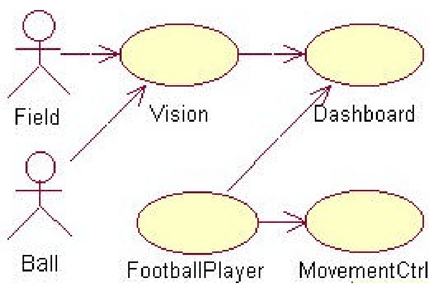


Fig. 3. Communications via a dashboard system

because it "shows actors and use-cases together with their relationships"[9]. We will show that, in our approach, use-cases at a certain level of abstraction can represent agents while actors can represent external entities (environment, other interacting software systems, users, …).

In order to illustrate our idea clearly, it is worth to declare the definitions of agent and of use-case that we have in mind:

"An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives"[1].

In the UML standard [9], about a use-case we can find that:

"A use case is a kind of classifier representing a coherent unit of functionality provided by a system, a subsystem, or a class as manifested by sequences of messages exchanged among the system and one or more outside interactors (called actors) together with actions performed by the system."

Using these two definitions we will try to establish a precise correspondence between use cases and agents and between the environment and the "outside interactors".

First of all, we have seen that an agent "is capable of flexible, autonomous action" then we can describe this through a use case that represents "a coherent unit of functionality". In so doing we represent the vocational behaviour of an agent with the use case illustrating his role.

Other important elements to deal with are agent's interactions; these interactions (with the real world and other agents) together with the agent design objectives, obviously, determine his own behaviours.

In the use case diagram these interactions can be related to the "messages exchanged". UML defines several kinds of standard relationships: association, extend, generalization, include. Other types can be user defined in order to fit the design needs.

With regards to the environment in which the agent is operating we can represent it as an actor that in UML *"defines a coherent set of roles that users of an entity can play when interacting with the entity"* [9]. In fact we can think that external stimuli give the agent the reason to perform a certain behaviour. Moreover we have already seen that the use case diagram definition refers to "one or more outside interactors (called actors)".

Now we could also identify as actor any agent who triggers some behaviour in another agent.

On this assumption however, we should proceed warily because about actors in the UML definition we can also find that *"Actors model parties outside an entity, such as a system, a subsystem, or a class, which interact with the entity…"* and therefore it's likely that this identification could be rightful (and useful) only in a specific moment of the life of the system (for example a scenario involving many use cases).

According to our experience, in agent software design it's often better to model agents internal to the system as use cases and all other external entities as actors.

This idea starts also from this sentence: *"Since an actor is outside the entity, its internal structure is not defined but only its external view as seen from the entity"*[9].

This argumentation brings to well known and interesting considerations about action and intention (see [10], [11]).

Representing agents as use cases, external entities and environment as actors and interactions among agents with relationships we can fully describe our system fom a functional, external point of view. We can also observe that this representation (from the implementation aspect) is:

a) structural because it gives evidence of the agents involved, of their relationships and of the type of relationships to be supported in the following steps (this is not different from the UML rigorous definition of the use case diagram because in our approach each use case that is still a functional aspect of the system represents an agent that will be implemented in the following steps becoming a structural element).
b) static because it doesn't clearly represent the dynamics of the interactions by which the agents achieve their scope (this description can be provided by the exploration of the possible scenarios);

B. Definition of the agents' structure

We will now define each agent as a class and will place it in a class diagram with his relationships with the other parts of the system.

In this step of the design we must be aware of the difference between agent based software and conventional object-oriented software.

This difference pertains not only to the discretional behaviour of an agent but particularly to his communicative possibilities.

In the previous phase we have spoken of some kind of relationships between the agents of our system: these relationships can be supported by the architecture that we

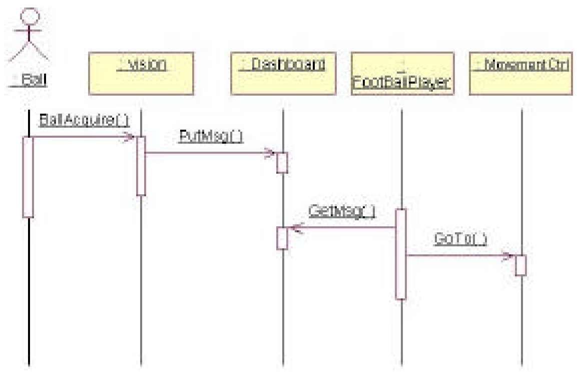


Fig. 4. An example of a scenario involving various agents

are going to establish in this phase. We should expect to find relationships supporting method invocations, event reactions, interactions among different agents [12].

Interactions are very important because can give place to the birth of an agent society. One of the most common way of implementing this feature is the publication/ subscription method. In the UML class diagram, two classes can be related through a "Subscribe" relationship that just denotes that "*objects of the source class (called the subscriber) will be notified when a particular event has occurred in objects of the target class (called the publisher)*".

Obviously it is also likely to find "use" or "communicate" relationships.

Note that in common object oriented programming, to each message corresponds one action (method invocation mechanism, cardinality one to one); this is a very poor situation if we want to implement an agent society.

Many agent environments (for example Ethnos: [22], [23]) support a one to many communication paradigm through the publication/ subscription method. Each agent subscribes the kind of messages he is interested in. When an agent has an information to communicate to the other agents who are interested, he publishes it and the others will read it from an interchange area (for example a dashboard system).

Looking at this architecture from a low level point of view we can see that each agent will perform a method invocation (the publisher to the dashboard and the reader also to the dashboard) but the cardinality of the communication is turned to one-many.

Consider the use case diagram of fig. 2 describing a simple football player robot.

In this schema the 'Vision' agent communicates to the 'FootballPlayer' through a simple method invocation. It calls a method to update the 'FootballPlayer' agent knowledge.

In the example of fig. 3, the same communication is performed via a dashboard.

Note that the communication starts from Vision and arrives to FootballPlayer. As already seen, this information exchange is performed through two method invocation: Vision use a post method of Dashboard and FootballPlayer use a read method of the same Dashboard.

If another agent (i.e. MovementCtrl) is interested in this message can read it too.

We can also discuss about the differences of this two types of communication from an execution time point of view but this is not the aim of this paper.

At this point indeed we have few elements to fix the internal structure of each agent. By now we can only create one class for each agent.

To complete the structure of each class we need a complete description of the scenarios depicting each use case (and therefore each agent behaviour). The importance of scenarios in requirements definition and modelling has been discussed by several authors (see [13] for a brief overview, other hints in [14], [15], [16]) and we will not go further on into this argument.

C. *Description of the behaviours*

The behaviour of the agents has to be captured in the various paths that can be identified in the use case diagram. In the description of these paths (scenarios) we will identify each agent's capability.

To proceed to the identification of a scenario we first have to formally define what is it. A scenario is "*a specific sequence of actions that illustrates behaviors. A scenario may be used to illustrate an interaction or the execution of a use case instance*".

Looking at this definition from the agent design point of view, in the diagram of fig. 3 we can identify and describe the following scenario:

"the Vision agent identifies the ball at a certain position and publishes a message to the dashboard. The FootballPlayer reads this message and decides to go towards the ball; then he orders to the movement control agent (MovementCtrl) to direct the robot towards a certain point."

Note that we have fixed a series of conditions, events and agents' choices generating this specific scenario. A different hypothesis has to be modelled in alternative scenarios: this also produces evidence for the nondeterministic nature of agent based software.

This scenario walks through several use cases and involves the participation of an actor. This is in accordance with the UML definition and is particular useful to describe the agent society interactions.

To design it we can use various diagrams: state/activity diagrams (showing the flow of processing, particularly useful to illustrate the concurrent execution of different tasks), sequence diagrams (showing the chronological series of communications), collaboration diagrams (showing the interactions between the agents/classes).

Sequence and collaboration diagrams are called 'interaction' diagrams because they show interactions between different parts of the system. In the sequence diagrams interactions are performed among instances through stimuli arranged in time sequence. Associations among these objects are not put into evidence in these diagrams: conversely, they are shown in the collaboration diagrams. In the following we will use the sequence diagram which is more indicated to deal with real-time systems for its time based ordering.

We can now illustrate the scenario previously seen by a sequence diagram (see fig.4).

In designing this diagram we use also the class diagram proceeding as described below. The agent Vision can post his message only if the agent Dashboard has the method

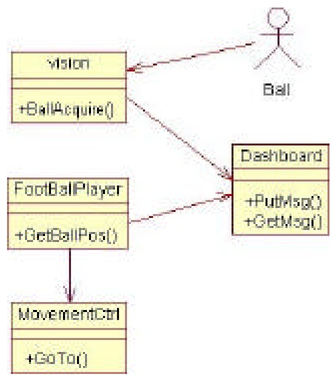


Fig. 5. The class diagram of the system

PutMsg. Then we have to:

a) create the two classes in the class diagram,
b) relate them with a relationship showing their interaction,
c) add the "PutMsg" method to the Dashboard class,
d) introduce the message "PutMsg" in the sequence diagram.

Both the structures of our agents (their methods and relationships as depicted in the class diagram) and the descriptions of their predicted behaviours grow together. This is one of the central ideas of our design process.

During this discussion we have thought had in our mind (but not talked about) the use of an UML CASE tool to design the software. At this stage, some requirements could be drawn for this tool too. To support this procedure the CASE tool should permit us to use method invocation as messages in the sequence diagram (this functionality is supported by several tools). The resulting class diagram is shown in fig. 5.

UML provides some possibilities for the software development process (entity, control, boundary, …) but none of them models the specific characteristics of an agent. For this reason we are used to define an "agent" stereotype.

What kind of stereotype is able to distinguish these classes? In the analysis phase three standard class stereotypes are provided by UML: entity (a passive class), control (it controls interactions between collections of objects) and boundary (it is on the boundary of the system and interacts with outside actors). No standard stereotypes are provided for design classes.

In our system, classes represent agents and in this perspective, it is not possible to model them as entity class. Our aim is to put into evidence that classes are active. For this reason we often define them with the user-defined 'agent' stereotype.

About control and boundary class we think that they should be deeper detailed by the 'agent' postfix (i.e. 'control agent', 'boundary agent').

Considering that Vision class is directly connected with an actor external to the system it could be considered as a 'boundary agent' class; note that its BallAcquire method is not really invoked by the Ball actor but it is his autonomous behaviour (probably invoked by the agent operating system).

In some specific application environment we can also further detail this stereotype introducing some time-related activation particular (for example: periodic, aperiodic, background).

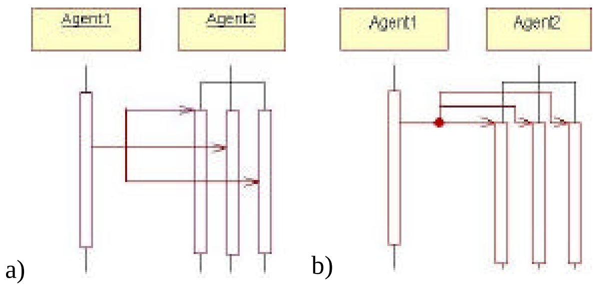


Fig. 6. Two different proposals for an extension of the sequence diagram notation

Supporting Concurrency

Agent based software often comprehend concurrent operations that are not yet supported by UML standard sequence diagram. The OMG Agent Work Group is still working upon this and several other proposals of extension to UML [18].

J. Odell proposes to extend the sequence diagram notation in order to support concurrent communications through various techniques [19]. The diagram in fig. 6,a shows a proposal of these.

In this example the first agent originates three different concurrent communications to the second agent who plays three different roles (or responds to three different communications). Other suggested extensions provide a decision box and an exclusive or.

This notation, in our opinion, is not coherent with the nature of a sequence diagram. In the definition of the UML sequence diagram we can found that: "A sequence diagram shows an interaction arranged in time sequence" and "… it shows the instances participating in the interaction … and the stimuli they exchange arranged in time sequence". We can deduce that throwing away the time axis from this diagram is not an extension but a radical modification of it. The three communications of fig. 6 can be concurrent only if the vertical time axis does not exist.

A way of introducing the concurrency, saving the nature of this context, could be done introducing a third dimension along which concurrent items could be represented maintaining their time sequence. We think that it is not difficult to create a suitable CASE tool supporting such a spatial diagram. The only big problem is representing on paper this diagram. Some solutions are possible (e.g.: perspective views for an overall sight of the diagram, sections and orthogonal projection in order to illustrate any detail) but a bi-dimensional representation is undoubtedly somehow difficult.

To avoid this problem though maintaining the nature of the sequence diagram we propose a different representation for concurrency.

Starting from a 'concurrency point' several messages can go towards different agents or agent's roles (see fig. 6,b). They reach the lifeline of each agent's role at the same height (i.e. the same time); this clearly illustrates the concurrent execution of the three different roles

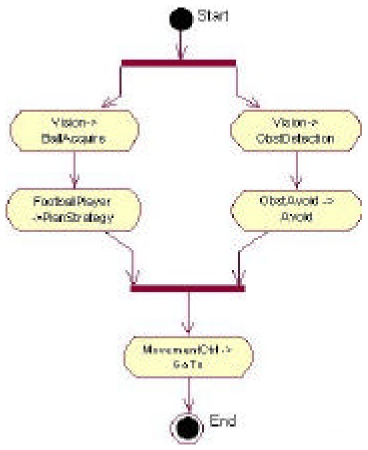


Fig. 7. Two agents concurrency in an activity diagram

In the previously seen scenario we have depicted a logical (and sequential) series of events but probably the agents will run in a concurrent environment in which several scenarios can take place at the same time.

Consider, for example, a second scenario in which an obstacle avoidance agent acts to avoid that the robot going towards the ball could hit something.

The FootballPlayer and ObstAvoid agents concurrently play their roles in order to achieve their scopes.

This situation can be depicted in an activity diagram (see fig. 7).

Note that in this diagram only 'behavioural' agents are depicted and as a consequence the communication to the dashboard is omitted.

Considering this way of using the activity diagram we can deduce another requirement for an UML CASE tool: it should be possible to introduce methods as elements of an activity diagram.

Iterating this process will lead to a complete design of the system. It is now necessary to create a component diagram to put each agent in an appropriate component and then deploy each component in its running unit (showing it with a deployment diagram).

V. FUTURE WORK

We are developing a CASE tool to support the AODPU. This tool will produce standard C++ code and will use an agent-software development library.

We are also making some efforts to study a way to the application of the ISO 9000-3 standard in the design of agent-oriented software. This will probably lead to a design process joining some aspects of the AODPU and the documents prescribed by the ISO rule.

REFERENCES

[1] J. Albus, H. McCain, and R. Lumia, "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)", NBS Technical Note 1235, Robot Systems Division, NIST, 1987.

[2] D. Lyons, and M. Arbib, "A Formal Model of Computation for sensory-Based Robotics", IEEE Trans. on Robotics and Automation, vol. 6, no. 3, pp. 280-293, 1989.

[3] L. Kaelbling and S. Rosenschein, "Action and Planning in Embedded Agents" in: P.Maes (ed.): Designing Autonomous Agents, MIT Press, Cambridge, MA, pp. 35-48, 1991.

[4] R. Brooks, "The Behavior Language", AI Memo 1227, MIT AI Laboratory, 1990.

[5] M. Wooldridge, N.R. Jennings and D. Kinny, "The Gaia methodology for agent-oriented analysis and designs", Internat. J. Autonoums Agents and Multi-Agent Systems 3(2000).

[6] N.R. Jennings, "On agent-based software engineering", Artificial Intelligence 117 (2000), 277-296

[7] I. Jacobson, G. Booch, J. Rumbaugh, "The Unified Process", IEEE Software, May/June 1999, pp. 96-102.

[8] F. Zambonelli, N.R. Jennings, and M. Wooldridge, "Organisational Abstractions for the Analysis and Design of Multi-Agent Systems", Proc. 1st Int. Workshop on Agent-Oriented Software Engineering, Limerick, Ireland, pp. 127-141.

[9] OMG Unified Modeling Language, version 1.3, June 99, Object Management Group document ad/99-06-08. Available from http://cgi.omg.org/docs/ad/99-06-08.pdf

[10] J. R. Searle, "Minds, brains and programs" in "The behavioural and Brain Sciences", Cambridge University Press, 1980.

[11] D.R.Hofstadter, D.C.Dennett: "The Mind's I", Basic Books, 1981.

[12] J. Odell, "Objects and Agents: how do they differ?". On-line at: www.jamesodell.com/publications.html

[13] M. Jarke, "Scenarios for modelling", Communications of the ACM, Jan 99, vol. 42-1, pp. 47-78.

[14] I. Jacobson, M. Christerson, P. Jonsson and G. Overgaard, "Object-Oriented Software Engineering: A Use Case Driven Approach", Addison-Wesley, 1992.

[15] A.G. Sutcliffe, N.A.M. Maiden, S Minocha, D. Manuel, "Supporting Scenario-Based Requirements Engineering", IEEE Trans. On Soft. Eng., Dec. 98, vol. 24-12, pp. 1072-1088.

[16] J.M. Caroll, M.B. Rosson, G. Chin Jr., J. Koenemann, "Requirements Development in Scenario-Based Design", IEEE Trans. On Soft. Eng., Dec. 98, vol. 24-12

[17] N.R. Jennings and M. Wooldridge, "Agent-Oriented Software Engineering", Handbook of Agent Technology (ed. J. Bradshaw) AAAI/MIT Press. (to appear).

[18] J. Odell, H. Van Dyke Parunak and Bernhard Bauer, "Extending UML for Agents", Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, Gerd Wagner, Yves Lesperance, and Eric Yu eds., Austin, TX, pp. 3-17, accepted paper, AOIS Worshop at AAAI 2000.

[19] J. Odell, H. Van Dyke Parunak and C. Bock. "Representing agent interaction protocols in UML". In OMG Document/ad99-12-01. Intellicorp Inc., December 1999.

[20] A. Chella, M. Cossentino, U. Lo Faso, "Applying UML use case diagrams to agents representation", Proc. of AI*IA 2000, Milano (Italy), 13-15 Sept. 2000, pp. 123-126.

[21] J.E. Robbins, N. Medvidovic, D.F. Redmiles, D.S. Rosenblum, "Integrating Architecture Description Languages with a Standard Design Method", II EDCS Cross Cluster Meeting in Austin, Texas, on-line at: www.ics.uci.edu/pub/arch/uml/research/.

[22] M. Piaggio and R. Zaccaria, "An Efficient Cognitive Architecture for Service Robots", The Journal of Intelligent Systems, Freund & Pettman, Endholmes Hall, England, Vol 9:2, March-April 1999.

[23] M. Piaggio and R. Zaccaria, "Distributing a Robotic System on a Network - the ETHNOS Approach", Advanced Robotics, The International Journal of the Robotics Society of Japan, Vol. 12, N.8, VSP Publisher, Aprile1998.