An agent based design process for cognitive architectures in robotics

Antonio Chella[(1)], Massimo Cossentino[(2)], Ignazio Infantino[(1)], Roberto Pirrone[(1)]
[(1)] DIAI, Università di Palermo, Palermo, Italy, [(2)] DIE, Università di Palermo, Palermo, Italy
chella@unipa.it, cossentino@unipa.it, infantino@csai.unipa.it, pirrone@unipa.it

Abstract

Nowadays, robots have to face very complex tasks, often requiring collaboration between several individuals. As a consequence, robotics can be considered one of the most suitable paradigms for agent-based software.
In this work, we present an approach to the design of distributed multi-agent architectures for mobile robotics, that is based on the Unified Modeling Language.
Our main goal is to provide a framework to perform a rigorous agent-based design process for this kind of systems both in the case of a single robot, and in a multi-robot scenario. Our methodology allows the designer to use system requirements in order to identify agents, and decide if they have to be implemented in a single mobile platform, or they're spread over many cooperative ones, or even shared by some repository.
Details of the methodology, system implementation using FIPA-OS environment, along with real and simulated experiments are reported.

1. Introduction

In recent years, mobile robots have been involved in more and more complex tasks often requiring the collaboration among several individuals that in general differ in their skills, and in the way they perceive the external environment.
From the architectural point of view, two different philosophies have been carried on: the reactive and the behaviour-based paradigms.
In this context, our work aims to propose a novel methodology for the design of multi-agent robotic architectures using the Unified Modeling Language.
Several other authors have used UML in the design of multi-agent systems. Some of them have discussed the possibility of using the UML to represent ontologies [20], role of agents and communications [18], others have proposed extensions of the UML to better deal with specific agent related problems like the agent interaction protocols [19], these efforts have produced various different proposal about AUML (Agent UML).
We have applied our methodology to the cognitive architecture previously developed by some of the authors, that could be viewed as an extension of the behaviour-based approach.
Particularly, the proposed methodology begins with the requirements analysis for the whole system, identifies agents, and defines behaviours also by means of classical FSA diagrams.
The agents defined in such a way are deployed on the required hardware platforms, thus allowing both single robot and multi-robot scenarios.
The paper is arranged as follows. Section 2 deals with the overall description of the multi-agent architecture; section 3 explains the design methodology; section 4 reports experimental results, while in section 5 some conclusions are drawn.

2. Description of the architecture

From the cognitive point of view, in our approach we refer to the architecture of fig. 1. In this structure it's possible to devise three main components: the perception, which is responsible to map the stream of raw data in an intermediate form, that in turn is provided to the cognitive component where the symbolic computation and, in general, deliberative behaviours of the system are located. The cognitive part can also support perception with some hints aimed to refine the perceptive process, and focus the attention on those external stimuli that are judged to be more useful for the current task completion. The third component is the actuation one, which communicates with the other two, in order to drive the robot hardware during perception tasks, and in attention focusing. The perception-action link allows also reactive behaviours.
Some of the authors already presented this architectural structure [14],[15],[16]. Its main goal is to go beyond the classical behaviour-based model, and to provide the robot with true "symbol grounding" capabilities due to the intermediate representation of sensory data, that is used to instantiate pieces of knowledge at the symbolic component. Through this mechanism the robot is able to act in a deliberative fashion more effectively.
The aim of this work is to provide a framework for our architecture allowing us to define a rigorous design methodology.
In particular, the scheme reported in figure 1 can be regarded as a categorisation of the possible agents typologies both if we look at the single robot

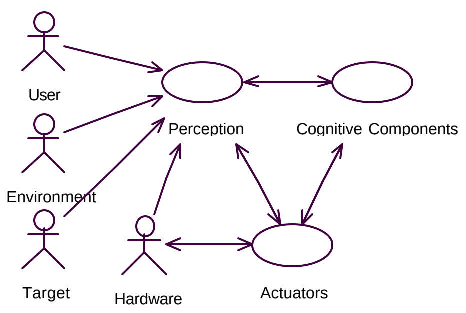

Figure 1: The architecture of a single robot from the cognitive point of view

architecture and if we consider a multi-robot scenario. In the second case we address the interaction between the external actors, and the whole team in order to perform cooperative tasks. In other words figure 1 is the highest level of abstraction in the system design, without taking into consideration the implementation details.
It's also possible to look at this issue as the holonic enterprise introduced by Brennan and Ulieru in [11].
The intra-enterprise level of a holonic enterprise can be matched to the architecture of fig.1. In fact, the entire system could be viewed as a distributed colony of agents interacting to achieve their own objectives; in so doing, they participate to accomplish the goal of the whole system.
In our point of view, the system can be viewed as a multi-robot, multi-agent structure. Each robot contains several agents; some of them interact with the external environment, while some other ones issue commands to the robot's hardware or they communicate with the agents of another robot.
Each agent is composed by a colony of tasks and plays a role that can be put into relation with one of the three areas reported in the general architecture of fig. 1. We suppose that there is a one-to-many relation between each one of these three areas and the agents of the system.

3. The design methodology

Design has been performed using an extension of the AODPU (Agent-Oriented Design Process with UML) methodology [7],[8] that is particularly useful in the case of robotic software architectures. The process is an iterative one, and it's described in fig. 2.
In a single iteration we could find the following phases:

- Requirements analysis
- Agents identification
- Definition of the agents' structure
- Description of the behaviours.

The "agent structure identification" and the "behaviour description" phases can be viewed as mutually dependent and cyclically performed to define the agents implementation. At the end of this process all the requirements are fixed (for this iteration) and the

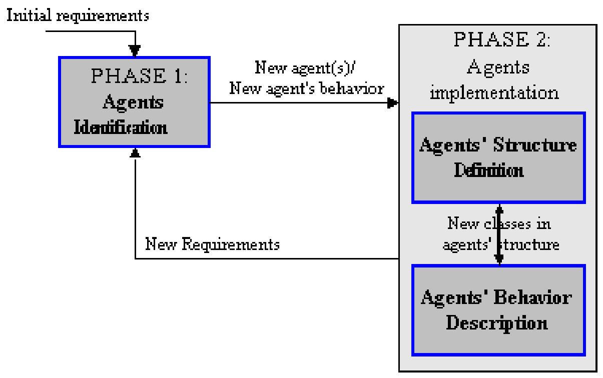
Initial requirements
PHASE 1: Agents Identification
New agent(s)/ New agent's behavior
PHASE 2: Agents implementation
Agents' Structure Definition
New classes in agents' structure
Agents' Behavior Description
New Requirements

Figure 2: The phases of the AODPU design process

implementation can start. In what follows, these design steps are described more in detail.

3.1 Requirements Analysis

A functional description of the system is provided through a hierarchical series of UML 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. In these diagrams the functions of the system will be formalized and in so doing it is also possible to use sequence diagrams in order to better illustrate the scenarios involved with the requirements.

3.2 Identification of the agents

Starting from the definition of an agent given by Jennings in [5] and looking at the definition of use case in the UML standard [6], we can prove that in a multi-agent system an use case can represent an agent and an actor can represent the environment [7].
It is also useful to consider that:

- the interactions between the agent and the world are a series of communication acts;
- the agent can achieve its scope through its own knowledge and functionalities (we suppose so);
- the knowledge of each agent can be increased using the communications with other agents or the real world;
- the behaviours of a single agent are implemented through a series of tasks.

It is important to underline that the relations among different agents in the use case diagram should be characterised by a "communicate" stereotype. Different kinds of relationships (for example "extend," "include" or "generalize") are not typical of a suitable agent identification schema; however they are also possible like in the case of direct method invocation to control device drivers (fig. 3). These communications will be

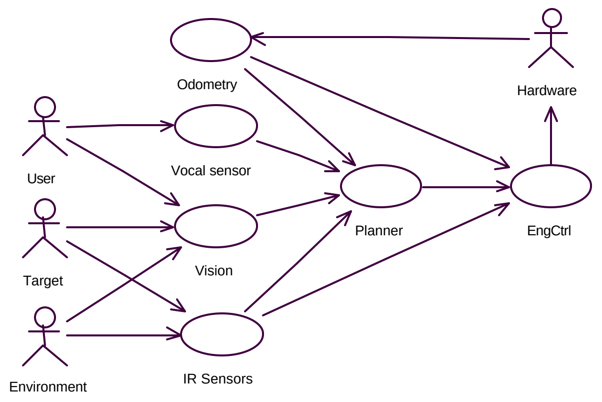

Figure 3: The use-case diagram for the identification of agents

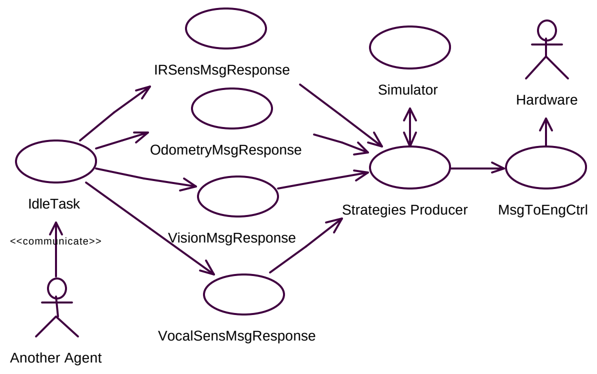

Figure 4: Detailed use case diagram for the "Planner" agent

obviously implemented through messages following a specifically designed ontology. Communications among (external) actors and the software agents are not implemented in such a way. The agents receive information from those actors through some sensing devices that are under their control.
This is a mediated communication whose ontology is constrained by the hardware devices and their drivers.
Detailing each use-case (which will be implemented as an agent) with another use-case diagram we can identify the tasks of the agent that are needed to perform the required functionalities. In this phase, it is possible to formalize the functionalities of the agent itself. With regards to the "Planner" agent in fig. 3 we could identify the tasks described in fig. 4.
The agent communicates with several other agents in order to receive information about the environment, the target and so on. For this reason a communication request handler ("IdleTask") is provided. It passes the communication to the specific handler for a particular message that will be responsible for it (for example the IRSensorMsgResponse handler is interested to the communications with the IR sensors agent). Using this strategy we could use a different ontology for the communication with each different agent.

3.3 Definition of the agents' structure

The next step of the process is the specification of the structure for each agent. Agents' structure can be provided through a class diagram. In order to perform experiments, we selected FIPA-OS [12],[13] as target architecture for implementation.
This choice is essentially based on the wide diffusion of this programming environment in the agent software community. Moreover, using Java ensures portability and a very efficient thread management. Inside FIPA-OS an agent is represented as a class, and it has to accomplish to several "tasks" regarding both communication with other agents, and performing its own duties. Tasks are represented as subclasses of the agent class. Following this approach, each use-case that has been identified as an agent is represented as a new class. The tasks of the agent are implemented through subclasses (one for each task).
With such a structure each agent can play its own role in the system organization using its own tasks (performed by the methods of its subclasses), knowledge (attribute of the agent class) and interactions with other agents (messages are sent/received by dedicated handlers tasks).
Tasks of a same agent interacts in a conventional object-oriented way without message exchanging but through methods invocation and attributes access [10] therefore a new task (being a subclass of the agent class) could be initialised or destroyed during runtime, each task could access the knowledge (attributes) of the agent it belongs and so on.
From the situation described in fig. 3 and fig. 4 it's possible to derive the structure depicted in fig. 5.

3.4 Description of the behaviours

It's possible to describe the scenarios relative to the use cases diagrams using some sequence diagrams: in 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 as in the case of mobile robotics. Similarly, one can describe the cooperation between agents in a particular scenario by the UML collaboration diagram.
In this phase we have to specify the behaviour of each task and the interactions between different tasks of the same agent or of different agents.
We can start looking at the behaviours of a single agent using the FSA approach of Arkin [9]. For the Planner agent of fig. 4 we can design the FSA diagram of fig. 6. The agent is in the "waiting" state until a message arrives. Then it turns to a state in which the message is processed, and the new information brings to the planning of a new strategy that enables execution by the actuators. The use cases of this agent as described in

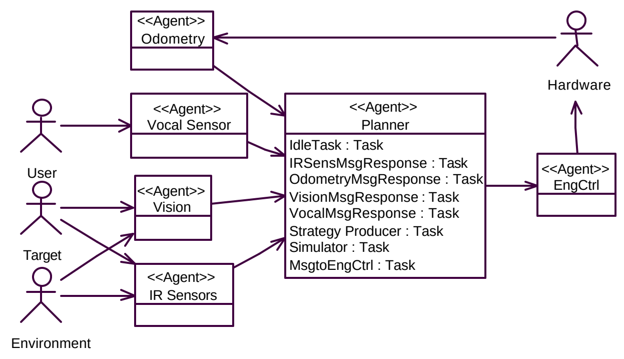

Figure 5: The class diagram illustrating the structure of the robot we developed for our experiments

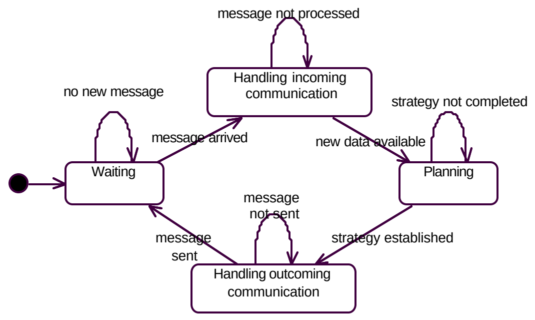

Figure 6: The behaviour of the "Planner" agent described with a FSA diagram

fig. 4 could easily be mapped onto the states of the FSA diagram. The "IdleTask" use-case actuates the behaviour described in the "waiting" task; when a message arrives, this task forwards it to the most suitable message-response task; this brings the system in the "Handling incoming communication" state.
When the information is extracted from the message and the required response procedure is completed the "Strategies Producer" initiates its work (state "Planning"). Once a new strategy has been defined the "MsgToEngCtrl" task is invoked to communicate it to the engine control agent (state "Handling outgoing communication").
Classical approaches to behaviours definition like the one reported in fig. 6 are mainly focused on single behaviours composition. When we use such a methodology we assume that a general framework is already present in order to manage the complexity of the whole system (subsumption, scheduling, dashboards and so on). Moreover, we are not able to derive any valuable information from this schema about implementation.
On the contrary, our design methodology allows us to explicitly detail relationships between robots, agents, or even behaviours. On the implementation side, we are able to derive not only the code structure in terms of class skeletons (see fig. 5) but the key elements of the methods' code.
The last issue can be addressed by the correspondence between the FSA-like diagram and a new UML activity one, describing the flow of methods invocations (fig. 7).
In this diagram each swim-lane is used to show the agent main class or a specific task. In the swim-lanes we put the methods of the correspondent agent/task class.
Between the previous FSA and this activity diagram we can establish some precise relations:

- the states of the FSA diagram correspond to one or more swim-lanes of the activity diagram.
- the transitions of the FSA diagram correspond to some of the transitions of the activity diagram in which the context related events are present.

For example in the "Planner.IdleTask" swim-lane we have the actions corresponding to the "waiting" state and the transition "message arrived" corresponds to the "new task (VisionMsgResponse)" invocation.

4. Experimentation

Experimental phase has been performed using both a software simulator and real robots. Simulator was needed to easily implement multi-robot scenarios. Two experiments have been set up: a prey-hunter competition using the simulator, and target reaching in the real case. In both experiments, robots were provided with obstacle avoidance capabilities.
All the implemented behaviours are quite simple because our study was mainly focused on testing architecture implementation rather than developing high quality solutions to accomplish the robot's tasks.
In particular, we were interested to stress multi-platform communication features of the FIPA-OS environment, and to cope with its lack of real-time control capabilities.
Our robot was a K-Team Koala equipped with IR sensors, and controlled by a PC through a serial link. Vision was provided by a calibrated camera looking at the action field, and reporting localisation information to the rest of the system. In order to test distribution of agents software across multiple platforms, the camera was connected to a separate PC running also the vision agent's code.
In what follows, a typical simulation experiment as long as the implementation of the vision agent will be reported in detail.

4.1 Prey-Hunter Simulation

Simulations were implemented using a Java GUI displaying a scaled action field, along with the obstacles layout. Robots are displayed using their 2D outline: in our simulation we used a Koala and a Kephera model. The Kephera plays the role of the prey and it is programmed to perform random trajectories.
It is to be noted that the only difference between simulation and real experiments is the implementation of

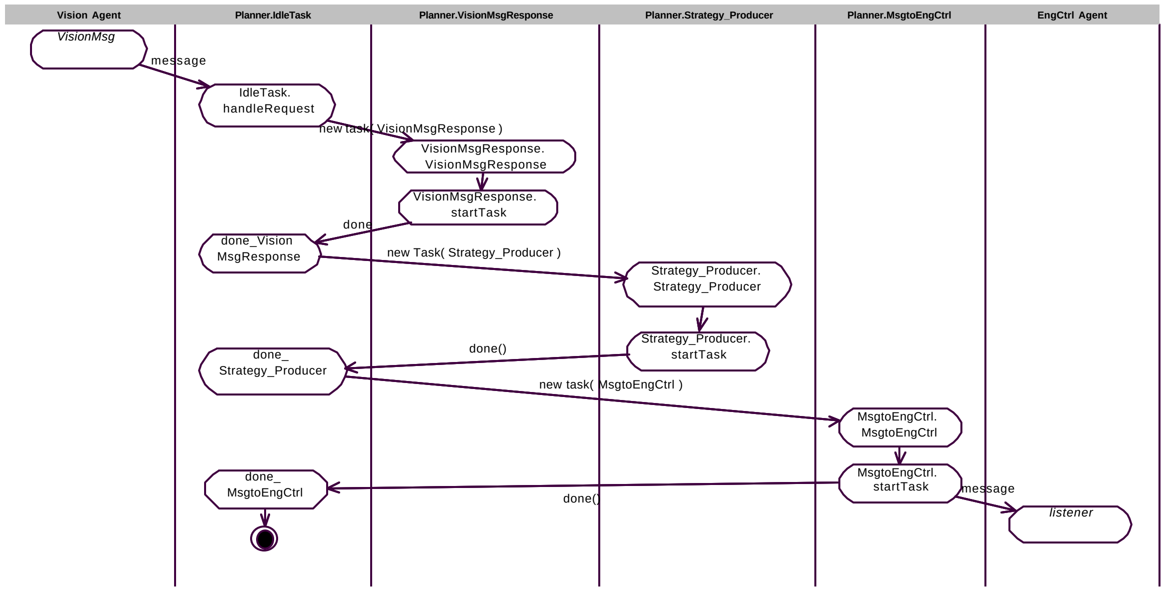
Vision Agent
Planner.IdleTask
Planner.VisionMsgResponse
Planner.Strategy_Producer
Planner.MsgtoEngCtrl
EngCtrl Agent
VisionMsg
message
IdleTask. handleRequest
new task(VisionMsgResponse)
VisionMsgResponse. VisionMsgResponse
VisionMsgResponse. startTask
done
done_Vision MsgResponse
new Task(Strategy_Producer)
Strategy_Producer. Strategy_Producer
Strategy_Producer. startTask
done()
done_ Strategy_Producer
new task(MsgtoEngCtrl)
MsgtoEngCtrl. MsgtoEngCtrl
MsgtoEngCtrl. startTask
message
listener
done_ MsgtoEngCtrl
done()

Figure 7: An activity diagram describing the details of the tasks' invocations

the core methods for the vision and motion control agents. Their communication interface with the rest of the system is the same as in real robot tests.
Simulation includes also time delays due to physical inertia. This approach allows us to make an extensive use of simulation in order to stress our design process.

4.2 Vision in Target Reaching

This section describes the process of localisation of the Koala robot during his task, in order to give useful feedbacks to the planning component [1].
We use a fixed CCD camera, connected to a computer, viewing the scene. The software component implemented can run on a different machine from that runs the rest of the system, communicating to it by socket over the local net. In this way we have the possibility of performing the vision task in real time without adding high computational costs to the whole system.
The valuable capabilities of the vision agent in the whole system are:

- to individuate and segment the Koala robot also in contrasted and irregular backgrounds;
- to perform a estimation of the position of the robot by camera images;
- to interpret the sequence of movements of the robot giving information of the direction followed by it.

The implemented computer vision task can be decomposed in three main steps:

- localisation of the robot on image by low-level image processing on the single frame;
- estimation of the 2D location of the robot;
- reconstruction of the 3D position of the robot .

4.2.1 Localisation of the robot on image. The position of the robot on image is calculated by simple low-level image processing operations. The current frame is subtracted to the previous (grey level images), obtaining the pixels related to moving objects in the viewed scene. If there are more than one object moving, Koala shape is selected using colour and texture features [2].
Naturally, some standard filtering operations are performed to reduce noise.
Moreover, a corner detector is applied in the area of the image representing Koala shape to have feature points to tracking. The estimation of the position of the robot on the floor is based on this tracked points.

4.2.2 Estimation of the 2D – 3D location. The position of the robot respect a reference system is estimated using the homography between image plane and floor [3],[4]. A generic 3D point X generates the point w on image:

$$l\hat{w} = P\hat{X} = K[R \mid t]\hat{X}$$

if the 3D points are on a plane (for instance Z=0), the transformation is simplified to a 3x3 matrix H:

$$l\hat{w} = H\hat{X}^{plane} = K\begin{bmatrix} r^1 & r^2 & t \end{bmatrix}\hat{X}^{plane}$$

where H is the homography matrix, decomposable on calibration 3x3 matrix K, and 3x3 matrix has the first two columns of the rotation matrix R and the translation vector

t. X and w are indicated using homogeneous coordinates. H is estimated using detected points belonging to the floor during a preliminary framework that also includes a calibration process: a grid placed in front of camera is used to obtain the calibration matrix K and fixes the rotation and translation referred to a reference system.
The tracked points on image are translated in 2D coordinates using estimated homography. The exact 3D position is recovered using the known real dimensions of the koala robot and the data coming from calibration framework.
The estimated 2D coordinates of the robot and the direction of the detected movements are communicate by a message to the system every time are calculated.

5. Conclusions

A novel methodology for the design of multi-agent robot architectures has been presented that extends the classical behaviour-based approach.
It has been showed that it can be profitably used both in the case of a single robot design, and in a multi-robot scenario.
The methodology has been implemented using a FIPA compliant, and the experimental results are very encouraging.

We're currently extending the methodology towards automatic code generation for a great part of the agents' implementation. In particular, regarding the agents' structure, we are looking at the classical categorisation reported by Russel and Norvig [17].

6. References

[1] Chella A., Gaglio S., Guarino M. D., Infantino I., "An artificial high-level vision agent for the interpretation of the operations of a robotic arm", Proc. 5th Int. Symp. on Artificial Intelligence, Robotics and Automation in Space, Noordwijk, 1999.

[2] Chella A., Di Gesù V., Infantino I., Intravaia D., Valenti C., "A Cooperating Strategy for object Recognition", in: R. Cipolla, D. Forsyth (eds): Proc. Of Int. Workshop on Shape, Contour and Grouping in Computer Vision (invited paper), Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1998.

[3] Faugeras O., "Three-Dimensional Computer Vision", MIT Press, Cambridge, MA, 1993.

[4] Horn B.P.K., "Robot Vision", MIT Press, Cambridge, MA, 1986.

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

[6] 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

[7] Chella A., Cossentino M., Lo Faso U., "Applying UML use case diagrams to agents representation", Convegno AI*IA 2000, Milan, Sept. 2000.

[8] Chella A., Cossentino M., Lo Faso U., "Designing agent-based systems with UML", IEEE International Symposium on Robotics and Automation, ISRA'2000, Monterrey, Mexico, Nov. 2000.

[9] Arkin R., "Behavior Based robotics", The MIT Press, Cambridge, Massachussets, London, England, 1998.

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

[11] Ulieru M., Walker S. S., Brennan R. W., "The holonic enterprise as a collaborative information ecosystem", to appear in Proc. of Autonomous Agents 2001 workshop "Holons: Autonomous and Cooperative Agents for Industry".

[12] O'Brien P., and Nicol R, FIPA – "Towards a Standard for Software Agents", in: BT Technology Journal, 16, (3), 1998, pp. 51-59.

[13] Poslad S., Buckle P., Hadingham R., "The FIPA-OS Agent Platform: Open Source for Open Standards", in: Proc. of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, UK, 2000, pp. 355-368.

[14] Chella A., Frixione M., Gaglio S., "Understanding dynamic scenes", Artificial intelligence, 123, 2000, pp. 89-132.

[15] Chella A., Gaglio S., Pirrone R., "Conceptual representations of actions for autonomous robots", Robotics and Autonomous Systems, 34, 2001, pp. 251-263.

[16] Chella A., Frixione M., Gaglio S., "An architecture for autonomous agents exploiting conceptual representations", Robotics and Autonomous Systems, 25, 1998, pp. 231-240.

[17] Russel S., Norvig P., "Artificial Intelligence: A Modern Approach", Prentice Hall Int. Ed., 1995.

[18] Bergenti F., Poggi A., "Exploiting UML in the design of multi –agent systems ", ESAW Worshop at ECAI 2000.

[19] Odell J., Parunak H., Bauer B., "Extending UML for agents", Proc. of the AOIS Worshop at AAAI 2000, Austin, TX, pp. 3-17, 2000.

[20] Cranefield S., Pruvis M., "UML as an ontology modelling language", in Proc. Of the Workshop on Intelligent Information Integration, 1999.