Using ABC^2 in the RoboCup Domain

Vicente Matellán, Daniel Borrajo, and Camino Fernndez

e-mail: {vmo,dborrajo,camino}@ia.uc3m.es
Departmento de Informática
Universidad Carlos III de Madrid
28911, Leganés, España

Abstract. This paper presents an architecture for the control of autonomus agents that allows explicit cooperation among them. The structure of the software agents controlling the robots is based on a general purpose multi-agent architecture based on a two level approach. One level is composed of reactive skills capable of achieving simple actions by their own. The other is based on an agenda used as an opportunistic planning mechanism to compound, activate and coordinate the basic skills. This agenda handles actions both from the internal goals of the robot or from other robots. This paper describes the work already accomplished, as well as the issues arising from the implementation of the architecture and its use in the RoboCup domain.

1 Introduction

The aim of this paper is to present the multi-agent architecture we are developing at University Carlos III.. This architecture (named ABC^2) is based on predefined *skills* that each robot composes in an opportunistic way to achieve an intelligent behavior. The way these basic actions can be combined to get more sophisticated behaviors is also pre-defined. This means that we are not using classical search-based planning to combine the actions. We use instead an agenda to keep a list of pending actions, where each action can require (or not) predefined simpler actions.

Actions can be inserted into the agenda by other actions, by events from the environment or by requests received from other robots. Similarly, actions can be accomplished as a result of the execution of other actions, by another robot actions or simply by changes in the world. Let us think, for instance, in an action of the RoboCup [4] domain, like *get-the-ball*. The robot can get the ball, either by its own actions (movements), asking another robot to pass the ball, or by any other event in the world (the opponent accidentally kicked the ball towards the robot).

For the definition of the skills, different types of controllers can be used, such as fuzzy controllers, mathematical calculus, or learned behaviors. For instance, we will use our previously developed software for building fuzzy behaviors. The cooperative part of this architecture is theoretically based on the Speech Acts theories [3].

In this area, our contribution will be in two aspects. First, we will extend these ideas to cope both with a highly dynamic environment (robotic soccer) and with a *real world* environment (in the sense of a sensors and actuators approach). Second, we will apply fuzzy logic both to define basic controllers and to write the heuristics that will control the robots.

We are not worried, by now, about any kind of learning, neither we do not try to figure out what the other team is trying to do, or doing. We also think that any kind of search-based planning is unuseful in such a highly dynamic environment.

In the next section the architecture is presented in more depth, discussing the skills that will be used. Section 3 describes the execution of the whole system, specially the role of the agenda in the control of the robot actions. The last section describes the current state of system development, and future work.

2 Description of the architecture

We have just introduced the goals of the architecture: allow explicit cooperation among the team mates, use opportunistic planning to combine robot actions, and use pre-defined basic reactive skills.

This section presents the general architecture we are developing, which is shown in Figure 1. This architecture is made up of different parts that will be explained in the following subsections.

2.1 Skills

The box labelled as *Skills* represents a set of simple and reactive controllers. These controllers implement pre-defined behaviors that the robot can accomplish. In Figure 1 some of these behaviors are shown.

The popularity reached by Rodney Brooks' work on the subsumption architecture [2] increased the interest in systems based on the composition of reactive *behaviors*. However, the idea of achieving intelligent behavior in robots using a bottom-up approach was not new. There had been many other work in the literature using the same approach, such as V. Braitenberg work [1]; neural networks based behaviors; or the most classical mathematical controllers. We have used fuzzy controllers to implement our behaviors. The main reason for this election was our previous experience using this controllers. We already had general fuzzy reasoning libraries so that we can easily design new controllers.

The design of the behaviors has been done heuristically. This means that we have chosen the rules by hand. However, many "automatic" methods for designing this type of rules can be found in the literature, ranging from the mathematical methods to neural networks or genetic algorithms. For instance, we have obtained good results using the last method [5]. However, most of these methods have been designed to learn in well-defined environments, with few dynamic objects, and they are highly time consuming. Besides, most of them do not bother about the multi-agent aspects of the soccer environment, though

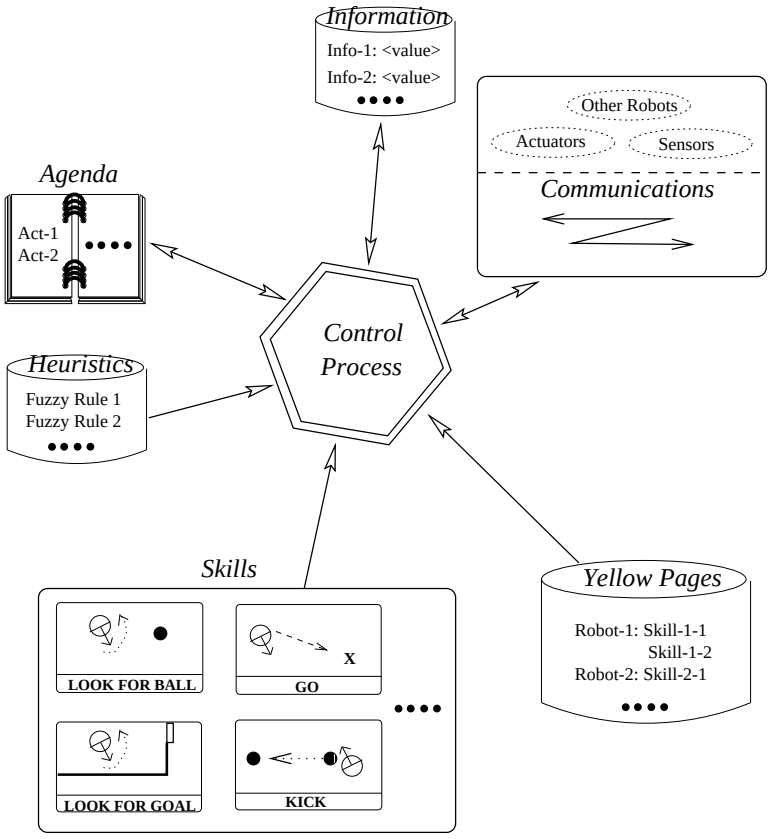

Fig. 1. Architecture of the Robots.

there are some new interesting work in this area [8]. We have decided to used hand-made fuzzy behaviors, because we can easily design them using high level rules, they perform well in the presence of uncertainty, and they can cope with multi-agent problems [6].

2.2 Yellow Pages

The repository named *Yellow Pages* in Figure 1 represents the information that an agent has about the other agents that form its team. This information basically consists of a table made by the name of its team mates, and the name of the skills they can accomplish. These skills will be used in the same way as its own skills.

A skill can be considered as an abstraction of an action that will be accessible to other team-mates. In fact, this means that we are considering that the robot

has meta-knowledge about itself (through its skills definition) and its team-mates (using the yellow pages).

2.3 Information

Classical reactive behaviors compute the outputs for the actuators of an agent directly from the raw numerical data perceived by its sensors. In other environments, like the RoboCup simulator, the inputs are not numerical data obtained from the sensors, but a mixture of linguistic and numerical information. In order to be able to handle this information we will use a reduced language that allows the agent to define the inputs of the skills and to keep significant information about the current state of the world. So, the skills previously defined use this language to represent the information used in the robot inputs. As an example, each agent keeps information about the distance and orientation to the goal.

2.4 Communications

One of the distinctive capabilities of agents is their ability to communicate with other agents. In order to be able to manage the intrinsic complexity of the communication (protocols, queues, etc.) we provide our agents with a specialized entity to cope with it.

Besides, in the RoboCup simulator [7], communication happens among two different agents, but also between an agent and its sensors and actuators. And both types of communication use the same channel in this case (a socket between the agent and the simulator). So, the entity in charge of the communication with the RoboCup simulator will have to be able to distinguish the different types of messages.

2.5 The Agenda

The *Agenda* is a dynamic structure that contains items named *acts*. These acts represent the potential actions that the robot is considering at a moment. We have considered four kinds of acts:

- `REQUESTED`, to indicate that the action in the argument of the act has been requested by another robot in order to be performed by this one.
- `REQUEST`, to ask another agent a particular action.
- `INFORMED`, presenting a piece of information sent by other robot.
- `SUPPLY_INFO`, to point out that some information has to be sent to another robot.
- `DO`, that represent potential skills that the robot can perform by itself. In the next section, the fundamentals of these behaviors are presented.

2.6 Heuristics

Heuristics decide at any time what act to select from the agenda. We have used fuzzy rules for the current implementation. The input variables of these rules can be, for instance:

- The priority of the skill associated to an act.
- The time that an act has been in the agenda.
- Number of acts that require an act to be evaluated.
- Information from the environment.
- The type of agent (defender, forward, etc.).

The output will be the weight of each act in the agenda. Once the acts have been weighted, the eligible act to be executed will be the one with the highest weight.

2.7 Agent Model

In summary, from the point of view of our model, the robot can be considered as a knowledge structure defined as a set of statical and dynamical attributes. Among the static ones we can quote the name of the robot (N), the list of its skills (S), the knowledge about its team-mates names and skills, (Y), the language to represent the information about the current state of the world (L), and the set of heuristic rules that governs the behavior of the agent (H). So, an agent (A) can be represented as the tuple: $A =< N, S, Y, L, H >$. In the same way, the team of agents can be represented as $< N, S, Y, L, H >^+$, given that a team is made up of at least one agent.

Among the dynamic information that defines the current situation of an agent, we can cite the agenda (A_g) that contains the acts currently under consideration, the queues of messages (Q) received or pending to be sent and the information (I) about the current state of the world, defined using the language L. So, an agent in a given moment is defined by $< A, A_g, I, Q >$, and the situation of the whole team as $< A, A_g, I, Q >^+$.

3 System Execution

The definition of a particular skill (top right box on figure 2) consists first on the design and implementation of the controller that performs the desired action (this is represented as the function *Execute* in the figure). Then, a condition for triggering the controller (named *Ready* in the figure) is established in order to know if the controller can be executed. In the case of the skill being evaluated but not being able to execute its associated controller (the *Ready* function returns a `FALSE` value), the skill provides a list of skills that can make it "executable". This list has been named *Needs* in the figure. The remaining slot of the box is the *Priority* assigned to the behavior. This value can be used in the heuristic rules to select acts from the agenda.

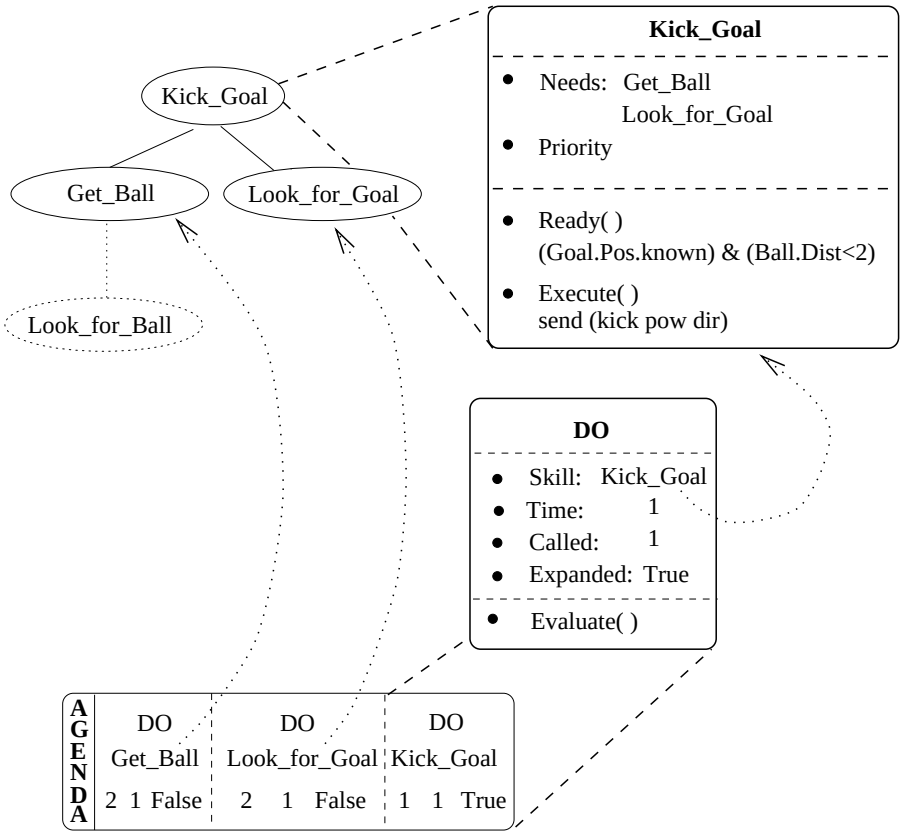

Fig. 2. Relations among the skills and the agenda.

Once the skills of the robot have been designed, (in this example we will consider only the four skills that appear in the top-left tree of figure 2), the heuristics have to be defined. Let us suppose that we settle up a simple heuristic: "Select from the agenda the act whose *Priority* value is the highest from the ones that are *Ready*". Let us also suppose that the information that the robot has about the world is the raw data that it gets from the RoboCup simulator and that it does not concern other team-mates skills. So, in this environment, we are considering only acts of type `DO`.

In such a simplified environment, the robot is only able to look for the ball and to kick it towards the opponent goal, according to the `Kick_Goal` skill. In order to do that, the robot has to be "stimulated" to do it. This means that the robot has to be initialized to pursue the goal `Kick_Goal`. This initialization is performed by inserting the act `[DO: Kick_Goal]` into the agenda.

The components of an act (as shown in Figure 2) are: the type of the act (`DO`, `REQUESTED`, etc.); the name of the associated skill; the counters `Called`,

that indicate the number of acts that require it, and `Time` that keeps the time when the act was inserted into the agenda; the switch `Expanded`, that indicates if the needs of the associated skill have been added to the agenda or not; and the function `Evaluate`, that indicates what has to be done when the act is selected (for example, execute its associated skill if the type of the act is `DO`).

The way the control cycle works is as follows: first, the applicable acts are selected. This is achieved by consulting the *Ready* function of the skill associated to each `DO` act. If the act is not applicable, then the `Expanded` switch is checked. If it has not been expanded its needs are inserted into the agenda like `[DO: <need>]` acts. This addition checks if that act had been previously added to the agenda by other acts. If the act already was in the agenda, the counter `Called` of the act is increased. Otherwise, a new act is added to the agenda. On the other hand, if the act is applicable and had been expanded, the counter `Called` is decreased. At the same time that the applicable acts are selected, the acts whose `Called` counter is equal to zero (no other act requires them) are removed from the agenda. Once the applicable acts have been selected, the domain heuristics are applied to select the one that will be evaluated.

The state of the agenda in Figure 2 shows that the act `[DO: Kick_Goal]` was inserted in the agenda at time 1 and it has been expanded. As a result of its expansion, the acts `[DO: Look_for_Goal]` and `[DO: Get_Ball]` were inserted at time 2. These acts have not been expanded, and are called by the act `[DO: Kick_Goal]`. This means that the agenda shows both the current state of the agent goals and part of the history of its activity.

The treatment of the other types of acts will be similar. Only the evaluation of these acts will be different. For instance, if an act `[REQUESTED: Look_for_Goal]` is evaluated, it can result in the evaluation of the skill `Look_for_Goal` and the insertion of its result as a `[SUPPLY_INFO: <result>]` act into the agenda.

4 Conclusion and Further Works

At Carlos III University we had been using preliminary versions of ABC^2 on our Khepera mini-robots [6]. As we had experience on designing reactive low level behaviors, we expected the main problem to be the design of the complex behaviors. So we began to work with the RoboCup simulator and the software we had previously built, which implements fuzzy controllers and the main part of the agenda-based control architecture, and we did not find many problems integrating both softwares.

The architecture was tested in the RoboCup'97 simulation track. The team lost its first match against CMUnited (9-1). It beat RMKnights (10-0) and lost against one of the teams of Kinki Universtiy.

The first conclusion we can get from these results is that if a team is better than another one in some particular tasks, then the results are really large. This is due to the fact that the competition is held on a simulator. So, we should try to focus on the issues that made a team better and not in the numerical results.

The first one was really a well tuned team. They had been working on this specific domain for a long time [8], and they had well-performing players and a nice global strategy. In summary, they actually have a good team. The only objection from our point of view is that they used a very specific approach. The second one, RMKnights, was also testing a general architecture in this domain. Its main drawback was that its players were too slow. We have less information about the third one, but we consider that the main reason for its victory was their control of the stamina parameter. This made them able to move faster in some periods of the match.

Now we are mainly working in two aspects of our architecture: Refining the basic behaviors and improving the cooperation mechanisms in order to use different attack strategies. In the first aspect we are studying, for instance, how to improve the basic skills using some kind of mathematical prediction about the robot moments (to know its position at any moment), the ball moments (to predict its position), etc.

References

1. Valentino Braitenberg. *Vehicles. Experiments in Synthetic Psychology.* MIT Press, Cambridge, MA (USA), 1984.
2. Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), 1986.
3. Philip R. Cohen and C. Raymond Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, RA-2(3):177–212, 1986.
4. Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the IJCAI-95 Workshop on Entertainment and AI/Life*, pages 19–24, 1995.
5. Vicente Matellán, José Manuel Molina, and Camino Fernández. Learning fuzzy behaviors for autonomous robots. In *Fourth European Workshop on Learning Robots*, Karlsruhe, Germany, December 1995.
6. Vicente Matellán, José Manuel Molina, and Lorenzo Sommaruga. Fuzzy cooperation of autonomous robots. In *Proceedings of the Fourth International Symposium on Intelligent Robotic Systems*, Lisboa, Portugal, July 1996.
7. Itsuki Noda. Soccer server: A simulator of robocup. In *Proceedings of AI Symposium'95*. Japanese Society for Artificial Intelligence, December 1995.
8. Peter Stone and Manuela Veloso. Towards collaborative and adversarial learning: A case study in robotic soccer. Technical report, School of Computer Science. CMU-CS-95-207. Carnegie Mellon University, 1995.