Journal of Intelligent and Robotic Systems **32:** 93–114, 2001.

ABC^2 an Agenda Based Multi-Agent Model for Robots Control and Cooperation

VICENTE MATELLÁN*
Departamento de Ciencias Experimentales e Ingeniería, Universidad Rey Juan Carlos de Madrid, 28933 Móstoles (Madrid), Spain;
e-mail: vmo@gsyc.escet.urjc.es

DANIEL BORRAJO
Departamento de Informática, Universidad Carlos III de Madrid, Avda. de la Universidad, 30. 28911 Leganés (Madrid), Spain;
e-mail: dborrajo@ia.uc3m.es

(Received: 20 June 2000; in final form: 31 October 2000)

Abstract. This paper presents a model for the control of autonomous robots that allows cooperation among them. The control structure is based on a general purpose multi-agent architecture using a hybrid approach made up by two levels. One level is composed of reactive skills capable of achieving simple actions by their own. The other one uses 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 two level approach allows the integration of real-time response of reactive systems needed for robot low-level behavior, with a classical high level planning component that permits a goal oriented behavior. The paper describes the architecture itself, and its use in three different domains, including real robots, as well as the issues arising from its adaptation to the RoboCup simulator domain.

Key words: agenda, control, cooperation, fuzzy, multi-agent, robots.

Abbreviations

ABC^2 – Agenda Based Cooperation for Agent's Behaviors Coordination
ADL – Agent Definition Language
AI – Artificial Intelligence
DAI – Distributed Artificial Intelligence
PRS – Procedural Reasoning System
UC3M – Universidad Carlos III de Madrid

* Vicente Matellán was lecturer at Universidad Carlos III de Madrid during most of the work reported in this article.

1. Introduction

The field of cognitive robotics, that is, robotic systems that can make their own decisions autonomous and intelligently, has evolved from the classic control theory applied to single robots with little control power, to Artificial Intelligence (AI) techniques applied to complex multi-robot domains such as RoboCup* (Kitano et al., 1995). The application of AI techniques transfered the debate on the merits of deliberative AI architectures (symbolic AI, cognitive AI, top-down AI or knowledge-based AI) versus alternative architectures (behavior-based AI, Artificial Life, bottom-up AI or reactive AI) to the robotics field. Among the first ones, the Shakey robot (Nilsson, 1984) is the best known. These systems used a classical AI view of planning and problem solving, which had, among other problems, the inability to reason and act on highly dynamic environments on real time. This is the case of most real-world problems that an autonomous intelligent system faces.

Among the research efforts on the second type of systems, the best known are the robots created by Brooks (1986) who focused the behavior of autonomous systems on using pre-defined "ad-hoc" behaviors (Connell, 1990). These architectures have been shown to be very effective on some domains and on specific tasks. However, more flexibility is strongly needed when designing a general architecture that requires high-level reasoning working with symbolic information and goals to be accomplished.

This paper presents a multi-agent architecture, named ABC^2 (**A**genda **B**ased **C**ooperation for **A**gent's **B**ehaviors **C**oordination) (Matellán, 1998), based on predefined *skills* (reactive component) that each agent composes in an opportunistic way to achieve an intelligent behavior (cognitive component). An agenda has been used 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 other robots actions, or simply by changes in the world.

This contribution extends this kind of system in two aspects. First, these ideas cope with highly *dynamic* and *real* environments and second, controllers can be designed using any reasoning paradigm including learned behaviors. This article presents three different domains where ABC^2 has been successfully tested (two using simulators and one real robots).

The paper is organized as follows. In the next section planning strategies most commonly used are compared in order to justify the design decision made in ABC^2. Section 3 formalizes the ABC^2 model, and its components. Section 4 thoroughly presents the first implementation based on the model and its application to the RoboCup challenge. This section also analyzes the results obtained in that competition. The last section describes an example of the execution of an implementation of ABC^2 model in real robots, focusing specially on the role of the agenda in the control of the robot actions.

* See `http://www.robocup.org` for more information about RoboCup competition.

2. Planning Strategies

When considering the application of a model for controlling robots to different tasks than the one it was specifically designed for, the adaptation requires hard work. This is so because each type of planning scheme is more appropriate to a type of problem due to its built-in assumptions. Cognitive AI models consider a plan to be a sequence of actions that have to be executed in order to achieve a goal. The traditional approaches to the automatic generation of these plans operate under assumptions such as (Fikes and Nilsson, 1971; Veloso et al., 1995; Penberthy and Weld, 1992):

- The state of the real world can be formally and correctly observed and defined prior to the search of a plan.
- The robot is the only agent that can modify the world.
- The robot actions have only the effects specified in its formal definition.

Under these assumptions, the control module (planner) is given a complete description of the initial state of the world, the potential actions (called operators) that the robot can perform, and a set of desired goals. The role of the planner will be to perform a search, usually exponential, expanding a tree of possible operators combinations to produce the sequence of robot actions (called plan) that leads from the initial situation to a situation on which the goals are fulfilled. Typically, these systems produce off-line complete plans. The whole plan is generated considering only the previously described assumptions.

There are other planning approaches with less strict assumptions, which include stochastic planners (García-Martínez and Borrajo, 2000), graph-based planners (Blum and Furst, 1995), or genetic programmed planners (Muslea, 1997).

A different paradigm, *reaction*, has been widely used in highly dynamical environments. This paradigm stands for intelligence resulting from the interaction with the environment, where the interaction is defined by small *behaviors* performing very simple tasks. The interaction in an autonomous robot is defined as a mapping from its sensors to its actuators. Planning in this context consists on several cycles of one-step decision making. One classical instance of this type of architectures is the *subsumption architecture*, where the key idea is using the world as its best model (Brooks, 1991). This approach divides the problem into activities, that connect sensing to acting directly. These behaviors must decide by themselves the moment when they have to act; they are not just subroutines to be invoked by a central planner but individual *behaviors*.

ABC^2 can be seen as an hybrid architecture made by two levels. The first one composed by skills that control the robot in a reactive way; and the second one uses a deliberative approach built as a high-level agenda-based planning mechanism that combines the skills.

Finally, the multi-agent aspect has also to be considered. The approach taken in ABC^2 has been influenced by different DAI (Distributed Artificial Intelligence) systems such as the one reported in (Bond and Gasser, 1988), and particularly by the

Coopera architecture (Sommaruga and Catenazzi, 1996). The cooperation mechanism is based on the Speech Acts theory (Cohen and Perrault, 1986), and a version of the Contract Net protocol (Smith, 1980) is used to assign tasks. The next section describes the model in detail.

3. Description of the Model

In this model, an intelligent system, in particular an intelligent autonomous robot, will be defined as a knowledge structure defined by a set of static and dynamic attributes. Among the static ones there is the name of the agent (N), the list of its skills (S), the knowledge about its team-mates names and skills, called yellow-pages (Y), the current state of the world, defined using a language (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:

$$A = \langle N, S, Y, L, H \rangle.$$

In the same way, given that a team is made up of at least one agent the team of agents can be represented as $\langle N, S, Y, L, H \rangle^+$.

For instance, let us consider a simplified team of robots, made up by two robots, `robot_1, and robot_2`. The definition of the first robot static components could be:

$$\begin{aligned}
\texttt{robot_1} = \langle\ & \texttt{robot}_1, \\
& \{\texttt{Go_Ball}, \texttt{Look_for_Ball}, \ldots, \mathit{skill}_{1n}\}, \\
& \{\texttt{robot}_2\texttt{: Pass}, \texttt{Go_Ball}, \ldots \mathit{skill}_{2m}\}, \\
& \{\ \mathit{Ball}\text{: (32,50)}, \ldots \mathit{Concept}_k\text{: } (\mathit{value}_{k_1}, \ldots \mathit{value}_{k_i}), \\
& \quad \mathit{Attacking}, \ldots \mathit{info}_j\ \} \\
& \{\ (\mathit{Ball}, \mathit{Goal}_{own}, \ldots, \mathit{Concept}_k)^+, \\
& \quad (\mathit{Attacking}, \mathit{Defending} \ldots \mathit{info}_l)^+\ \}, \\
& \mathit{heuristic\text{-}rules}\ \rangle
\end{aligned}$$

where $skill_{1j}$ (j from 1 to n) represent the n available skills of `robot_1`. For example $skill_{11}$ may be `Go_Ball`. Then, the information that `robot_1` has about the m skills of `robot_2` is represented, given the names of the skills of that robot (for instance, *Pass*). Then the description of the environment is given. This information is described using the language L, and usually will be organized into *structured information*, composed by a classical hierarchy of concepts, such as the *Ball*, and its attributes, for example its position; and *unstructured information*, such as if the team is attacking or defending. Finally, the set of heuristic rules for controlling the agenda is given.

Among the dynamic information that defines the current situation of an agent there is 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

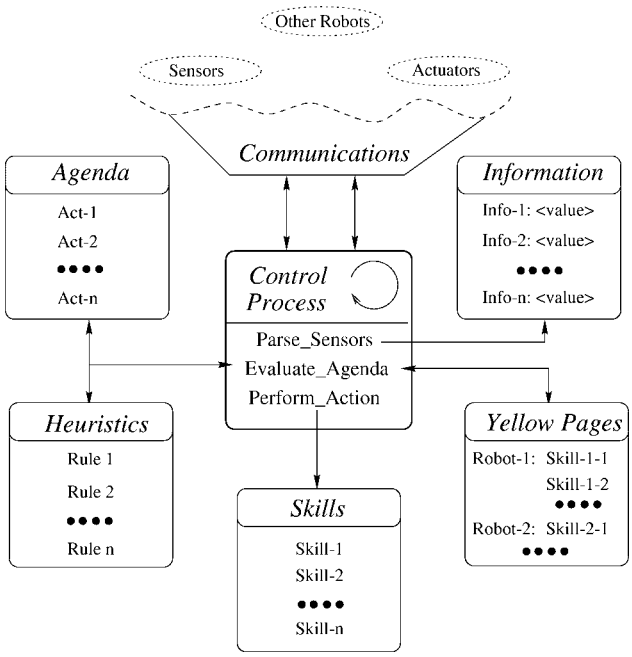

Figure 1. Architecture of the robots.

in a given moment is defined by $\langle A_g, I, Q\rangle$, and the situation of the whole team as $\langle A, A_g, I, Q\rangle^+$, where A is the name of the agent representing each tuple. This definition is graphically shown in Figure 1.

In a given instant, the situation of both robots could be defined as:

$$\begin{aligned}
\langle robot_1, \quad & A_g = \{\ [\texttt{DO:Go_Ball}]\ \},\\
& I = \{\text{Ball.position=}(20, 90), \mathit{Attacking} = \mathit{True}\},\\
& Q = [\texttt{REQUESTED:Go_Ball}, robot_2]\ \rangle\\
\langle robot_2, \quad & A_g = \{[act_1] \ldots [act_n]\},\\
& I = \{obj_1.attribute_1, ..., obj_k.attribute_n\},\\
& Q = \text{none}\ \rangle
\end{aligned}$$

which means that `robot_1` has only one act in the agenda. This act is to perform the skill `DO:Go_Ball`. The only information that `robot_1` has about the elements of the environment is that the Ball is at distance 20 with angle 90, and that its team is attacking, which corresponds to the structured and unstructured information given using language L. Finally, the queue of messages shows that the `robot_2` has asked `robot_1` to perform the skill `Go_Ball`.

3.1. DESCRIPTION OF COMPONENTS

The following description explains in more detail the components of the model shown in Figure 1:

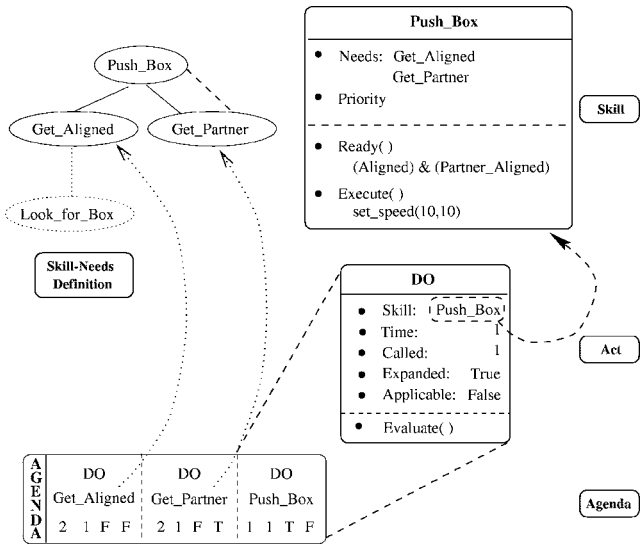

Figure 2. Relations among the skills and the agenda.

Skills Set of simple and reactive controllers. These controllers implement predefined behaviors that an individual agent can accomplish. They can be implemented using any type of decision-making mechanism or learned ability. In particular, for the example described in Section 5, fuzzy controllers (Zimmermann, 1990) have been used. The main reason for this election was that fuzzy controllers are very flexible, which makes them easily adaptative. Besides, fuzzy sets theory is a well-suited paradigm that has shown its effectiveness in many autonomous systems. Also, fuzzy theories are susceptible to be shared among different agents (Matellán et al., 1996). And finally, we had the possibility of using previously developed and tested fuzzy reasoning libraries (Matellán et al., 1995a).

The design of the behaviors has been done heuristically for the examples of the applications on next sections. This means that the rules have been chosen by hand. However, many "automatic" methods for designing this type of robot behaviors can be found in the literature, ranging from the mathematical methods (Steels, 1990) to neural networks (Maes and Brooks, 1990) or genetic algorithms (Koza, 1991). For instance, good results have been obtained in previous work using the last method (Matellán et al., 1995b). However, most of these methods have been designed to learn in well-defined environments, with few dynamic objects.

The definition of a particular skill (as shown on the top right box of Figure 2) requires:

– Setting the condition for triggering the controller (named *Ready* in the figure) in order to know if the controller can be executed or not. This condition is equivalent to the requirement in deliberative planning that,

in order to apply an operator, its preconditions have to be true in the state.
- The design and implementation of the controller that performs the desired action (this is represented as the function *Execute* in the figure), whose effect is equivalent to the representation of the post-conditions of classical planning operators.
- Providing a list of skills that can make it "executable". In the case that the *Ready* function of a skill returns a `FALSE` value, it cannot be immediately executed. In order to make it executable, one has to define other skills that, after their execution, allow the previous skill to be executed. This list of skills has been named *Needs*. This list simplifies the search of potential operators that provides the preconditions in deliberative planners.
- Establishing the *Priority* assigned to the behavior. This value can be used in the heuristic rules to select acts from the agenda and reflects the "a priori" importance of that skill.

The Agenda The *Agenda* is a dynamic structure that contains items named *acts*, whose name is due to the Speech Acts theory (Cohen and Perrault, 1986). These acts represent the potential actions that the robot is considering at a given moment. Five types of acts are considered in *ABC*^{2}:

`DO <skill>`, that represent potential skills that the robot can perform by itself. In the next section, the fundamentals of these behaviors are presented.

`REQUEST <agent> <skill>`, to ask another agent to perform a particular skill.

`REQUESTED <skill> <agent>`, to indicate that the skill in the argument of the act has been requested by another robot in order to be performed by this one.

`SUPPLY_INFO <agent> <info>`, to point out that some information has to be sent to another robot.

`INFORMED <info> <agent>`, to get a piece of information sent by another robot.

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 counter `Called` that indicates the number of acts in the agenda that require it;
- The counter `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;
- 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`).

Section 3.2 explains in more detail how the agenda works.

Heuristics They decide at any time what act to select from the agenda. Fuzzy rules have been used in the implementations described in this article. We are currently exploring other types of heuristic representations for comparison purposes. The input variables of these rules are, among others: the priority of the skill associated to an act; the time that an act has been in the agenda; the number of acts that require an act to be evaluated; the information from the environment; and the type of agent.
The output is the weight of each act in the agenda. Once the acts have been weighted, the eligible act to be executed is the one with the highest weight. For example, an heuristic rule could be "**if** *Attacking* **is** True **and** *Distance(Ball)* **is** Near **then** `Go_Ball`.*Priority* is *Increased*". These heuristics can also be used to purge the agenda of undesired acts.

Yellow Pages Knowledge that an agent has about the other agents. This information 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 the robot has meta-knowledge about itself (through its skills definition) and its team-mates (using the yellow pages).

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 for instance the RoboCup simulator (Kitano et al., 1995), 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, a reduced high-level language, named L is used. It 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 use this language to represent the information of the robot inputs.

Communication One of the distinctive capabilities of the agents controlled by ABC^2 is their ability to communicate with other agents. In order to be able to handle the intrinsic complexity of the communication (protocols, queues, etc.) the agents are given a specialized entity to cope with it.

This entity provides the abstraction needed to cope with different communication environments. The abstraction gives a unique mechanism of communication of the type:

```
send(Message, Destination)
```

The way this procedure is implemented depends on the type of problem. For example, in a simulator the function is implemented over the TCP/IP tower, while in a real robot it uses the available communication mechanism. So, in the RoboCup simulator (Noda, 1995), UDP sockets have been used while in the real example shown in Section 5 serial port communication has been used. This abstraction is the lowest level of the cooperation mechanism whose main component is the Speech Acts previously described.

3.2. CONTROL CYCLE

Previous section describes the components of the control mechanism of the robot. This one presents the control cycle that controls each agent. This control cycle mainly consist in the following control algorithm:

Initialize (*Agenda*)
while $Agenda \neq \emptyset$
  Recursively *Remove* Act_i such as $Act_i.Called = 0$
  $Acts_{applicable} = \{Act_i \in Agenda$ such that Act_i.Ready = True $\}$
  Repeat $\forall Act_i \in Agenda$ such that $\notin Acts_{applicable}$ **loop**
    if not $Skill_{Act_i}$.Expanded **then** Expand $Skill_{Act_i}$
  $Act \Longleftarrow$ Select *Act* from $Acts_{applicable}$ using *Heuristics*
  Evaluate (*Act*)

where *Agenda* is the agenda of the robot, $Acts_{applicable}$ the subset of the acts contained in the agenda whose associated skill is *Ready*, `Initialize` inserts the initial act into the *Agenda* and `Empty` checks if the *Agenda* is empty or not.

The way this algorithm works is as follows: first the *Agenda* of each agent has to be initialized in order to achieve any particular task. This is performed by inserting an initial act into its agenda. For instance, a `DO` act with a skill requiring high attention. This act can be considered as its main goal or its initial goal. Other acts will be generated as they will be required in order to achieve this initial goal, for instance as needs of this act (see Figure 2).

Another way acts can be inserted into the agenda, apart from their insertion as needs of other acts, is directly by the `Execute()` function of a skill that can indicate the addition of another act to the agenda.

Then, the applicable acts of the agenda are selected. This is achieved by consulting the *Applicable* feature of the act. The way this feature is calculated depends on the type of act. For example, in a DO act is set from the value of the *Ready* function of its associated skill. If a DO act is not applicable, then the Expanded switch of its skill is checked. If it has not been expanded, its needs are inserted into the agenda as [DO: <need>] acts. When adding acts to the agenda it checks if the considered act had been previously added to the agenda by other acts. If the act was already in the agenda, the counter Called of the act is increased; otherwise, a new act is added to the agenda.

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 and the counter Called of all its needs that were in the agenda are decreased. This is repeated recursively until there is no modification neither in the number of acts into the agenda nor in the values of the Called counters.

Once the applicable acts have been selected, the domain heuristics are applied to select the one that will be evaluated. The application of the heuristic rules results in a selected act. Finally, the selected act is evaluated. If the selected act is a DO act, it executes the *skill* associated to that act. If the selected act was a REQUESTED, it inserts a new DO act* containing the requested skill into the agenda, etc.

This control cycle continues while the agenda contains any act. That is, the control cycle of the agent exits (and the agent itself) while it has something to do. If an agent with unlimited life is needed, it is only needed an act DO whose associated skill has a null *Execute* controller and a *Ready* function that will never be true.

The main advantage of this approach over the other two extremes, deliberation and reaction, is that it can flexibly vary between the two. On the deliberative extreme, one could always introduce new acts into the agenda, until all acts had been expanded. Then, one would select one applicable skill and deliberate again, or execute another skill. On the other extreme, one could always prefer to execute skills as they become executable, becoming a reactive system. But, here, we have another alternative that allows us to switch between both extremes becoming more opportunistic with respect to the the defined heuristics, and the current state of the agent and the environment.

4. ABC^2 in the RoboCup Environment

RoboCup (Kitano et al., 1995) is a challenging competition both of real and simulated robots. The goal is to test architectures that can control a team of robots in a multi-agent, real-time and noisy environment. In the simulator competition, where the model was tested, each team was made by eleven players, each one controlled

* This is so in this implementation of ABC^2, REQUESTED acts are translated into DO acts, which means that the agent treats the accepted requests from other agents in the same way that its own action, but it is being considered if a direct execution of the skill will be more adequate.

individually. The robots get unpredictable noisy sensory information in real-time, and have to take an action each 100 ms. The communication among the members of the team was unreliable, unsecure, broadcasted (messages from the opponent team were also received) and with a low-bandwidth.

ABC^2 was used directly (without any adaptation apart from the design of the specific skills) to this environment. Each robot was controlled by its own implementation of the model, using its own subset of skills, set of heuristic rules and initial act.

4.1. DEFINITION OF AGENT'S SKILLS

The following is the summarized description of some of the skills used in our team (Matellán et al., 1998). This is a reduced and simplified version of the skills used in the competition, provided only to give an overview of the model and the type of skills to be designed.

- Look_for_Ball:

  Ready: Ball position is not known.

  Execute: The robot turns 85 degrees right.*

- Go_Ball:

  Ready: Ball position is known and the distance to the ball is bigger than 2 meters.

  Execute: The robot turns and dashes in the appropriate direction.

- Kick_Goal:

  Ready: Distance to ball is less than 2 meters (kickable), opponents goal direction is known.**

  Execute: Kicks at maximum speed (100) towards goal. The position of obstacles in the way towards the goal is considered to calculate the direction.

- Go_Position:

  Ready: The player knows its own position and it is not in its initially assigned position. Of course, the decision whether the player is in its assigned position or not has a tolerance to avoid unnecessary movements.

  Execute: Go to its initially assigned position.

- Pass:

  Ready: Distance to ball is less than 2 meters (kickable). Team-mate position its known.‡

* In the real implementation there is a controlled randomness that prevents from turning always in the same direction and avoids alternative turns right-left.

** Decision of kicking towards goal or not is not considered in the skill.

‡ In the real implementation there are different kinds of Pass skills that allow defensive and offensive strategies. Also the ability to pass towards vacant spaces has been considered.

Execute: Kicks towards team-mate position using a calculated strength. The position of opponents in the way towards the team-mate is considered to calculate the direction.

- Look_Ball_at_Position:

  Ready: The player is in its assigned position, ball position is known and it is not looking straight towards ball.

  Execute: Turns towards ball.

- Come_Out:

  Ready: Ball position is known and the distance to it lower than 15 meters.

  Execute: Turns and dashes towards the ball.*

- Clear:

  Ready: Ball distance is less than 2 meters (kickable).

  Execute: Kicks the ball forward avoiding opponents.

- Look_for_Team_mate:

  Ready: Always.

  Execute: Turns 90 degrees towards right.

- Advance_towards_Goal:

  Ready: Ball distance is lower than 2 meters (kickable).

  Execute: Kicks the ball towards goal and dashes in that direction.**

- Win_Match:

  Ready: The time is over.

  Execute: The result of the game is written and player program ends.

These skills may have been heuristically designed or may have been learned. Both mechanisms for designing can be mixed in the same agent with no restriction. In the real implementation, skills are C++ classes derived from a base class. So, a learned behavior has to be implemented in this way, overloading functions `Ready` and `Execute` of the class.

4.2. DEFINITION OF THE PLAYERS

Once the skills have been defined, the players are designed. One player consists, as it was defined in Section 3.1, in the definition of its skills, the relations among them, the heuristics, and an initialization. In order to make the definition of agents easier in different domains, *ABC*^{2} provides an Agents Definition Language (ADL).

The definition of any player using ADL is divided into the following parts:

* If the ball is moving, a prediction is used of to where it is going to be.

** The full implementation considers other skills implementing different advanced strategies.

1. Initial parameters.
2. Skills and needs declaration.
3. Initial skill.
4. Information about other agents.
5. The definition of the heuristics.

In order to show how ADL is used, let us present the configuration of an agent of the RoboCup domain. For instance, a simple goal keeper (goalie). Its desired behavior will be to stay in its goal, remain in its position looking at the ball and try to catch the ball if it approaches closer than 15 meters. Then, if he catches the ball it will try to clear it. The definition of this player will be made as follows:

```
* Initial parameters
-50 0 0 3 goal keeper
* Skills
Go_Position 0.7
Look_for_Ball 0.6
Keep_Looking_at_Ball 0.75  Go_Position Look_for_Ball
Get_Out 0.8  Keep_Looking_at_Ball
Kick_off 0.9  Get_Out
Win_Match 1  Kick_off
* Initial Skill
Win_Match
* Skills of team-mates
Left_Defender: Kick_off, Pass, Receive, ...
...
* Heuristic definition
goalie.heuristics
* End_of_File
```

Lines starting by $*$ are comments and they separate the different parts. The first part sets the initial position in the field: (X,Y) coordinates, orientation and tolerance in that position, as well as the name of the player, where its position ($X = -50,\ Y = 0$) corresponds to the center of its own goal.

Then, the skills that the robot can use are defined. For each skill an initial weight (*Priority*, in Figure 2) is set, as well as its list of needs. For instance, the skill `Keep_Looking_at_Ball` has two needs `Go_Position` and `Look_for_Ball` and an initial weight of 0.75. Every need of a skill has to be previously defined, and it has to correspond with an appropriate implementation (providing the *Ready*, *Execute*, etc. functions).

Once the skills of the robot have been defined, it is the turn of the heuristics. In the version of *ABC*^{2} used in the RoboCup fuzzy rules were used to implement the heuristics. In order to make them easily adjustable, the heuristics were defined in a separate and human-readable format (ASCII file). The definition of the heuristic

rules means the definition of the inputs and outputs of the rules, and also the specification of the heuristic rules themselves.

In the implementation used in the RoboCup, first the name of the inputs are given, for instance `DB` (Distance_to_Ball). Then the linguistic labels (such as **V**ery **N**ear, **N**ear, **F**ar and **V**ery **F**ar) used to qualify that variable are defined. The definition of these labels is made by specifying their membership functions (Zadeh, 1973) as trapezoids. For example, **V**ery **N**ear (**VN**) is defined by -1, 0, 2, 2.1 in the following example. These four values correspond to the abscissas coordinates of a trapezoid of height 1 defined over the range of the input `DB`. The range of the input (Distance_to_Ball) obviously should be positive, since it is a distance, however the first vertex is negative (-1) to put the slope in the negative area, so the height of the trapezoid would be 1 (that is, crisp $VN = 1$) when Distance_to_Ball would be zero.

Then, the outputs are defined in the same way, that is, by giving the four abscissas coordinates of a trapezoid. Output variables will represent the modification of the *Priority* of the different skills. For example, the *Priority* of the skill `GO` (**G**etting **O**ut of its goal) can be (**D**ecreased, kept **E**qual or **I**ncreased).

Finally, the heuristic rules themselves are defined as first order rules made by a condition (that can use the fuzzy operator *AND*, written as &), the implication symbol `=>`, and the result (where the fuzzy operator *AND* can also be used).

The following statement corresponds to a small part of the definition of the heuristics for the goalie (`goalie.heuristics`), and the definitions previously described are shown all together:

```
* Inputs
{ DB
VN -1 0 2 2.1
N 0 2 10 15
F 8 15 20 25
VF 22 30 120 130 }
...
* Outputs
{ GO
D -2 -1.2 -0.8 0
E -0.5 -0.2 0.2 0.5
I 2 1.2 0.8 0 }
...
* Rules
{
if DB is VF => GO is D
if DB is F & DG is ... => GO is D & ....
... }
```

The design of the heuristics have been made heuristically attending to the experience in previous matches, but learning methods could be used to improve them,

because they are defined in a high level language. We will explore that possibility in future work.

The real behavior implemented for the goalie used some other skills (trying for example, not only to clear the ball, but to pass it towards a team-mate, etc.) and more sophisticated heuristics.

4.3. RESULTS

The first implementation of the ABC^2 model was tested in the RoboCup'97 simulation track. There were 33 participants in this track (U.S. = 8, Europe = 8, Australia = 2, Japan = 15). Teams were organized in groups of 4 teams. Our team team lost (1–9) its first match against CMUnited, team presented by Carnegie Mellon University (USA). It beat RMKnights (10–0), the team presented by the Royal Melbourne Institute of Technology (Australia), and lost (0–8) against Niken, team presented by Kinki University (Japan). In summary, two matches lost, one won, 17 goals received and 11 scored.

This was the first RoboCup competition, and some considerations have to be taken into account when evaluating matches results. The first one is that if a team is better than another one in a particular task, then the difference of goals in a match may be really large (results in other groups were in the order of thirty goals of difference between the two teams). This is due to the fact that the competition was held on a simulator, which enforces these goal differences. Most of the time, one special issue (good or bad) eclipses the rest of the characteristics of the team. So, an analysis of the performance of a team should try to focus on the issues that made a team better and not in the difference of goals.

In our case, this was also our first entry into this kind of competition, and we realized that some of our skills were not well tuned. ABC^2 intends to provide a framework for the coordination of robot skills, but it does not deal with the skills themselves. For example, the prediction of the ball movements from the information received from the simulator can be considered just a mathematical issue. However, this is a key element when evaluating a player performance. For instance, the goal-keeper actions heavily depend on the ball movement prediction. Instead, we decided to focus on implementing the action selection mechanism proposed in ABC^2, than on implementing a perfect ball movement prediction. In the same way, for this particular issue we have preferred to try learning techniques (Fernández et al., 1999), than just developing mathematical stuff for improving this behavior.

The second one is that the competition itself is not the overall goal of RoboSoccer. The main goal of RoboSoccer is to show that different control architectures and models can be successfully applied to a highly uncertain and dynamic environment. Therefore, we did not focus on the performance of the behaviors, but on the use of the ABC^2 model. From this point of view, we succeeded because ABC^2 could be used without requiring any modification.

5. Experiments with Real Robots

Let us consider an example where two robots have to push a box at the same time from the same side. In order to simplify the problem, we defined two points where the robots have to be aligned to adequately push. This experiment has been tested both in a simulated environment (see Figure 3), and also using two real robots (see Figure 4). In the simulated one we have used the simulator SimDAI (Sommaruga et al., 1996) built in our lab. This simulator lets users define independent processes implementing each one of the agents. In the figure two windows

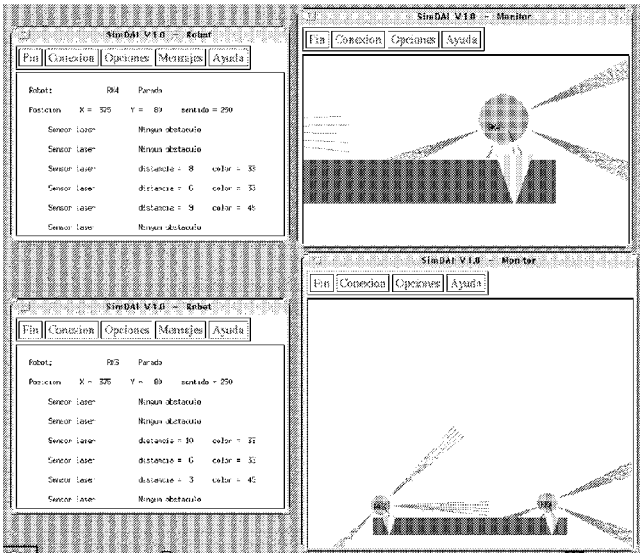

Figure 3. An example in a simulated environment.

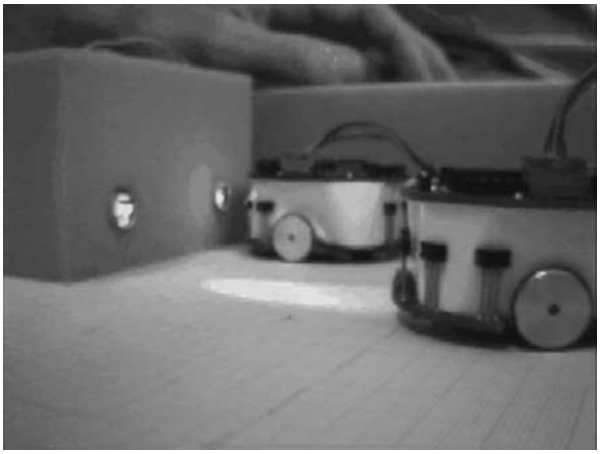

Figure 4. The real environment.

for monitoring both of the robots are shown (in the left), as well as two windows to monitor how the system is working (the upper right window shows a detail of the simulated environment and the lower right the whole system).

The real robots we have used are Khepera micro-robots (Mondada et al., 1993). These robots have only infrared sensors, which makes difficult getting information about the object to be pushed. In order to make the alignment points reachable to the robots, these points correspond with two lights placed inside the box to push in the real environment of Figure 4. A skill makes the robot push when it is aligned (`Push_Box`). Another forces it to be aligned (`Get_Aligned`). A third skill looks for the alignment point (`Look_for_Box`), and the fourth one asks the other robot for getting aligned (`Get_Partner`).

Once the skills of the robots have been designed (in this example only the four skills enumerated in the previous paragraph), the heuristics have to be defined. Let us suppose that a simple heuristic is settled up as: "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 received by its sensors (that will be translated into definitions about the distance to the object, the angle, the obstacles, etc.) and the information about whether its team-mate is aligned or not, that will be received as `INFORMED` acts from its team-mate.

In order to achieve the task of having the robots push the box, they should be initialized. This is performed by inserting the act `[DO: Push_Box]` into the agenda. Then, the applicable acts are selected. This is the situation reflected in Figure 2 where the act `[DO: Push_Box]` was not applicable and it has been expanded by inserting the acts `[DO: Get_Aligned]` and `[DO: Get_Partner]` into the agenda.

The execution of the skill `Get_Partner` may result in the insertion in the agenda of an act such as `[REQUEST: RobotB, Get_Aligned]`. In order to know what skills other robots can perform, it consults its Yellow Pages which describe for the rest of agents what skills they can execute. The evaluation of this act would result in sending this request to the other robot. The treatment of the other types of acts is similar. Only the `Evaluate()` method (see Figure 1) of these acts is different. For instance, if the act `[REQUESTED: Get_Aligned, RobotA]` is evaluated by the RobotB, it would produce the insertion of the act `[DO:Get_Aligned]` in its agenda.*

In a similar way, the evaluation of the skill `Get_Aligned` would cause the insertion of a `[SUPPLY_INFO: RobotA, Aligned]` act into the agenda. The evaluation of this one will generate an act with the information: `[INFORMED: Aligned, RobotB]` into the agenda of the first robot, and its evaluation in toggling the predicate `Partner_Aligned`. When the two robots realize that they are aligned and get the information that their partner also is, they can push together. In a similar way they can handle the amount of time that they have to push, the strength to apply, etc.

* Or not, if the heuristics of the second robot decide, for example, that acts containing requests from RobotA are discarded.

6. Related Work

Section 2 introduced the two classic approaches to robot control: deliberative and reactive. ABC^2 was defined there as an hybrid architecture based on opportunistic planning. The characteristics of these approaches can be summarized as in Table I.

Hybrid architectures intend to combine reactive and deliberative control, and usually consist of three components: a reactive layer, a planner, and a layer that links the other two. Well known examples of this kind of architecture are AuRA (Arkin and Balch, 1997), which integrates a A^* planner with schema-based controllers (Arkin, 1989), and PRS (Procedural Reasoning System) (Georgeff and Lansky, 1987) based on least commitment via plan elaboration postponement.

Opportunistic architectures (as ABC^2) are a subset of the hybrid architectures that take its name from Barbara Hayes-Roth approach to hybrid control (Hayes-Roth, 1992). The agent architecture on her system was also made up by three components: an event-triggered reactive level, an strategic planner, and a control process in charge of matching triggered actions with the generated plan. A similar architecture is used in O-Plan (Currie and Tate, 1991) where the term "agent" is used to name each of the three modules of the system.

A closer evolution of these ideas, from the ABC^2 view point, are RAPs (Reactive Action Packages) proposed by Firby (1996). RAPs were designed to allow the reactive execution of symbolic plans. In this way, each RAP defines different alternatives of execution depending on the environment, and an agenda is used to select the next action to execute. Another approach, also close to ABC^2, is the TCA (Task Control Architecture) by Simmons et al. (1997), which integrates symbolic plans with real-time restrictions as well as reactive behaviors triggered as exceptions. Both architectures, RAP and TCA, use similar approaches to the hybrid control of robots to the one used in ABC^2. However, none of them takes into account the multi-agent problems that are one of the key elements, and the most significant contribution of the ABC^2 architecture.

Once the foundations of ABC^2 have been presented, the remaining question to clearly situate ABC^2 in the robotics control literature is to determinate the types of robotic problems for which each control method is more appropriate. Table II classifies (+ for the best, − for the worst, and ∼ for intermediate) previous planning alternatives according to some features of robotics problems.

Table I. Summary of planning architectures

Model	Representation	Assumptions	Goals
Deliberative AI	First order logic	Static world	Flexible, explicit
Opportunistic planning	Propositions, Skills	Pre-planned actions	Flexible, implicit
Reactive planning	No world model	Markov	Fixed, implicit

Table II. Relation among planning approaches and problem features

Problem Feature	Deliberative	Reactive	Opportunistic
TYPE OF ENVIRONMENT			
Uncertainty in Operators	−	+	∼
Uncertainty in Information	∼	−	+
Explicit Goals	+	−	∼
Representation of State	+	−	∼
TIME REQUIREMENTS			
Speed	−	+	∼
Dynamic changes in environment	−	∼	+
Design effort	+	−	∼
MULTI-AGENT CAPABILITIES			
Communication abilities	∼	−	+
Coordination	−	∼	+
Cooperation	+	−	+

These features can be classified into three main groups: the type of environment, time requirements and the number of agents involved. The type of environment includes both the characteristics of the environment itself (if it is structured, how dynamic it is, etc.) and the relevant features the architecture is based on. It is clear that in highly dynamic environments (where the assumption that the agent is the only one that can modify the environment does not hold) deliberative planners are not useful. They keep replanning continuously. It is also well known that reactive systems, where there is no abstract representation of the world, neither of the goals, can be hardly used in high level tasks.

Another important feature to be considered is the design effort needed to adapt the system to a specific environment. Deliberative systems use high level representation (operators, predicates, etc.) and they are easily adapted. On the other hand, reactive systems are built as dedicated processes and are designed "ad hoc" for each specific application. This means that a reactive system usually has to be completely rebuilt for every application. Opportunistic systems, such as ABC^2, are in the middle. Skills are designed for specific tasks but the way they are combined can be easily adapted to different domains. Also, skills themselves can be reused.

With respect to time requirements, if the tasks to accomplish have to be decided in milliseconds (for instance to turn in front of an obstacle) the system can not perform a huge search. So, most deliberative AI planners are discarded in "real-time" environments. On the other hand, if the time is not critical, deliberative planners are able to find a better solution than the reactive ones, that may not leave from local minima. Of course, opportunistic planning also has its problems, most

of them based in the fact that skills concatenation is fixed. This allows very fast search for a solution, but there is no way to look for a new concatenation if the fixed one can not be applied.

Finally, evaluating the influence of multi-agent domains means establishing the degree of cooperation among the agents. A first level is communication, where agents only share information. In this level agents usually need to share a language (which implies internal representation) to exchange information. The second level is coordination, which means that agents actions are executed without interferences; that is, one agent actions do not block or invalidate other agent actions. This coordination can be achieved using explicit or implicit communication (reactive systems). The third level implies that the actions of the agents are organized in order to maximize the performance of the whole group and usually requires shared goals, explicit coordination and communication.

According to Table II, opportunistic planning would be the most appropriate one to be used in the three different domains presented in this paper, where there are at least two agents: two simulated robots in the first domain; two real robots in the second; and eleven players in the RoboCup environment. Also, these environments are dynamic and unpredictable.

7. Conclusions and Further Work

The experiment using real robots was designed first in the simulator, where it was easier to debug the skills, and then they were used in the real robots. We used the same skills in both environments (except for slight differences in the treatment of the sensorial information). This meant that the architecture and algorithms were general enough that they could easily be applied and parameterized to different environments. Of course, in both environments the robots were able to push the box cooperatively. (you can reach some demos at `http://gsyc.escet.urjc.es/~vmo/videos`).

Summarizing, in this paper we have described the ABC^2 model for robots control and cooperation. Its fundamentals have been explained and also the theoretical principles that have influenced it. Then, two applications have been presented in order to show how the system works in different domains solving different robotic tasks. The experiments have shown that this architecture is well suited to domains where there is a need of great flexibility in the accomplishment of the skills, that is, environments where opportunistic planning can be used. Besides, it allows an intuitive method to deal with cooperation among agents by letting agents define their own skills, and the rest of the group having knowledge of them. We have also shown how this architecture adapts to different environments by the definition of the particular skills, the relation among them and the heuristics to control their execution. Preliminary results show that the model is also suitable for real robots applications: it is fast enough, easily traceable, and it can be first designed in a simulator and then moved into the real platform.

We are currently studying how to improve some of the basic skills, such as *intercepting a moving ball*. We are taking two approaches, the first one is using numeric prediction about the robot moments (to know its position at any time), the ball moments (to predict its position), etc. The second approach consists of using reinforcement learning techniques (Fernández et al., 1999) to acquire the behavior of primitive skills.

References

Arkin, R. C.: Motor schema based mobile robot navigation, *J. Robotics Res.* **8**(4) (1989), 92–112.

Arkin, R. C. and Balch, T. R.: AuRA: principles and practice in review, *J. Experimental and Theoretical Artificial Intelligence* **9**(2) (1997).

Blum, A. L. and Furst, M. L.: Fast planning through planning graph analysis, in: C. S. Mellish (ed.), *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95*, Vol. 2, Montreal (Canada), 1995, pp. 1636–1642.

Bond, A. H. and Gasser, L.: *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann, 1988.

Brooks, R. A.: A roboust layered control system for a mobile robot, *IEEE J. Robotics Automat.* **RA-2**(1) (1986), 14–23.

Brooks, R. A.: Intelligence without representation, *Artificial Intelligence* **47** (1991), 139–159.

Cohen, P. R. and Perrault, C. R.: Elements of a plan-based theory of speech acts, *Cognitive Sci.* **RA-2**(3) (1986), 177–212.

Connell, J. H.: *Minimalist Mobile Robotics: A Colony-style Architecture for a Mobile Robot*, Academic Press, Cambridge, MA, 1990.

Currie, K. and Tate, A.: O-plan: The open planning architecture, *Artificial Intelligence* **52**(1) (1991), 49–86.

Fernández, F., Borrajo, D., and Matellán, V.: VQQL: A model to generalize in reinforcement learning, in: *Proceedings of the European Conference on Planning*, Durham (UK), 1999, pp. 385–387.

Fikes, R. E. and Nilsson, N. J.: STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence* **2** (1971), 189–208.

Firby, J. R.: Modularity issues in reactive planning, in: *Proceedings of the Third International Conference on AI Planning Systems*, Edinburgh (UK), 1996, pp. 78–85.

García-Martínez, R. and Borrajo, D.: An integrated approach of learning, planning, and execution, *J. Intelligent Robotic Systems* **29**(1) (2000), 47–78.

Georgeff, M. P. and Lansky, A. L.: Reactive reasoning and planning, in: *Proceedings of AAAI-87 Sixth National Conference on Artificial Intelligence*, Seattle, WA (USA), 1987, pp. 677–680.

Hayes-Roth, B.: Opportunistic control of action in intelligent agents, *IEEE Trans. Systems, Man, Cybernet.* **23**(6) (1992), 1575–1586.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E.: Robocup: The robot world cup initiative, in: *Proceedings of the IJCAI-95 Workshop on Entertainment and AI/Life*, 1995.

Koza, J. R.: Evolving emergent wall following robotic behavior using the genetic programming paradigm, in: F. Varela and P. Bourgine (eds), *Toward a Practice of Autonomus Systems. Proceedings of the First European Conference on Artificial Life*, Cambridge, MA, 1991, pp. 110–119.

Maes, P. and Brooks, R.: Learning to coordinate behaviors, in: *Proccedings of the Eighth National Conference on Artificial Intelligence*, San Mateo, CA, 1990, pp. 796–802.

Matellán, V.: *ABC^2: An architecture for intelligent autonomous systems*, PhD Thesis, Dept. Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, 1998.

Matellán, V., Borrajo, D., and Fernández, C.: Using ABC^2 in the RoboCup domain, in: H. Kitano (ed.), *RoboCup-97: Robot Soccer World Cup I*, Lecture Notes in Artificial Intelligence, 1998, pp. 475–483.

Matellán, V., Molina, J. M., and Fernández, C.: Fusion of fuzzy behaviors for autonomous robots, in: *Proceedings of the Third International Symposium on Intelligent Robotic Systems*, Pisa, Italy, 1995, pp. 157–164.

Matellán, V., Molina, J. M., and Fernández, C.: Learning fuzzy behaviors for autonomous robots, in: *Fourth European Workshop on Learning Robots*, Karlsruhe, Germany, 1995, pp. 45–50.

Matellán, V., Molina, J. M., and Sommaruga, L.: Fuzzy cooperation of autonomous robots, in: *Proceedings of the Fourth International Symposium on Intelligent Robotic Systems*, Lisboa, Portugal, 1996.

Mondada, F., Franzi, E., and Ienne, P.: Mobile robot miniaturisation: A tool for investigation in contr ol algorithms, in: *Proceedings of the Third International Symposium on Experimental Robotics*, Kyoto, Japan, 1993.

Muslea, I.: SINERGY: A linear planner based on genetic programming, in: Sam Steel (ed.), *Recent Advances in AI Planning. 4th European Conference on Planning, ECP'97*, Toulouse, France, 1997, pp. 312–324.

Nilsson, N. J.: Shakey the robot, Technical Report 323, SRI A.I. Center, 1984.

Noda, I.: Soccer server: A simulator of RoboCup, in: *Proceedings of AI Symposium'95*, 1995.

Penberthy, J. S. and Weld, D. S.: UCPOP: A sound, complete, partial order planner for ADL, in: *Proceedings of KR-92*, 1992, pp. 103–114.

Simmons, R., Goodwin, R., Zita, K., Koening, S., and O'Sullivan, J.: A layered architecture for office delivery robots, in: L. W. Johnson (ed.), *Proceedings of First International Conference on Autonomous Agents*, Marina del Rey, California (USA), 1997, pp. 245–252.

Smith, R. G.: The contract net protocol: High-level communication and control in a distributed problem solver, *IEEE Trans. Comput.* **C-29**(12) (1980), 1104–1113.

Sommaruga, L. and Catenazzi, N.: From practice to theory in designing autonomous agents, in: *First Australian Workshop on Distributed Artificial Intelligence*, Lectures Notes in Artif. Intell. 1087, Springer-Verlag, 1996, pp. 130–143.

Sommaruga, L., Merino, I., Matellán, V., and Molina, J. M.: A distributed simulator for intelligent autonomous robots, in: *Proceedings of the Fourth International Symposium on Intelligent Robotic Systems*, Lisbon (Portugal), 1996, pp. 393–399.

Steels, L.: Cooperation between distributed agents through self-organization, in: Jean-Pierre Muller (ed.), *Descentralized A.I.*, Elsevier Science, 1990.

Veloso, M., Carbonell, J., Pérez, A., Borrajo, D., Fink, E., and Blythe, J.: Integrating planning and learning: The PRODIGY architecture, *J. Experiment. Theoret. Artif. Intell.* **7** (1995), 81–120.

Zadeh, L. A.: Outline of a new approach to the analysis of complex systems and decision processes, *IEEE Trans. Systems, Man, Cybernet.* **1** (1973).

Zimmermann, H.-J.: *Fuzzy Sets. Theory and its Application*, Kluwer Acad. Publ., Boston, MA, 1990.