Combining Classical and Reactive Planning: the ABC^2 Model

Vicente Matellán **Daniel Borrajo**
e-mail: {vmo,dborrajo}@ia.uc3m.es
Departamento de Informática. Universidad Carlos III de Madrid
C/ Butarque 15. 28911, Leganés (Madrid), España

Abstract

Systems that must plan, act, and probably learn using a rational, flexible behavior in real time present a challenge to current integrated schemas of planning and acting. Classical planning is still far from solving problems on such short time as needed in dynamic domains, such as Robosoccer, even with new approaches such as graph-based techniques. Reactive systems on the other hand, are hard to configure and do not work well when high level flexible reasoning is required.This paper presents an opportunistic planning-acting schema based on our model named ABC^2. The primary goal of the model is to compose pre-defined *skills* in an opportunistic way to achieve and intelligent behavior for robots working autonomously.

Introduction

The design of intelligent controllers for autonomous robots (known as intelligent robots) has been an attractive research area for many years. In a first stage, controllers used a cybernetic approach (Wiener 1948), where controllers were based on the classical negative feedback loop of control. Some of them achieved interesting results considering the stage of the electronic technologies in that moment, such as Walter's turtle (Walter 1953). These types of controllers were used till the generalization of digital computers and the emergence of Artificial Intelligence (AI) as the main paradigm used for controlling robots.

Since the beginning of AI, two basic trends have driven the research in this field: the cognitive and the reactive approaches. This paper summarizes both "schools" in order to compare their planning and execution approaches with the one we proposed in ABC^2 (**A**genda **B**ased **C**ontrol for **A**gents **B**ehaviors **C**oordination). In this way, a brief historical review of their influence in the robotics field is presented. During the seventies, most approaches were based on the classical AI paradigms (abstraction, planning, heuristic search, etc.) (Nilsson 1984). These type of controllers were based on the advances achieved in the automatic generation of plans. Systems mainly used the large digital computers available in those days. They operated under assumptions such as (Fikes & Nilsson 1971):

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

In the last eighties, there was a strong criticism (Brooks 1991) to this approach, mainly based on its limitations to cope with real world uncertainties and dynamics. They claimed that intelligence results from the interaction with the environment, which gave the name to the trend: "reaction". In this way, autonomous robots intelligence will be a result of its interaction with the real world through its sensors and actuators. Systems built using these ideas, such as the Herbert robot (Connell 1990), were able to deal with the real world successfully.

These ideas are not new. In fact, they are a new release of the cybernetic approach, that in turn, was influenced by dominant ideas in the psychology during the first decades of this century (Pavlov 1927). These ideas, based only on the stimulus-response approach, were rejected by cognitive psychology (Neisser 1967) using similar arguments to the ones used nowadays by researchers using classical planners. The main one is that the incoherence between locally decided actions and final goals, can cause the robot never be able to achieve its global goals. Another one is their lack of flexibility; systems are built for an special application, and they are hardly reusable. Both are very related to the fact that high level goals cannot be explicitly formulated in a reactive system. Only low-level goals are considered, and their formulation may not be very explicit; it depends on the underlying paradigm used to implement the behaviors. The only way high level goals are achieved in this kind of systems is by combining the low level behaviors in a prefixed and invariant way. For instance, in the *subsumption architecture* (Brooks 1986) this combination is made by describing an activation/inhibition network of behaviors.

The problem of controlling a group of robots naturally extends research on single robot control. So, the two main AI trends have their own reflection in the field of multiagent systems. The one based in reac-

tive approaches is known as "collective behavior" (McFarland & Bsser 1993) where intelligent behavior arises out of the interaction among not very capable agents and it is the result of innate behavior of the individuals. The other one, based on traditional (symbolic) AI paradigms, is named "cooperative behavior" and it is based on the intentional desire to cooperate in order to increase individual utility. ABC^2 model was designed to cope with multiagent problems and we took this last approach. We supposed that our model will provide selfish agents able to cooperate in order to face common problems.

Nowadays, the dominant trend in planning systems applied to control is to build hybrid systems. On one hand the cognitive approach tries to deal with the uncertainties using statistical models, such as (Blythe 1996). On the other hand, reactive systems help high-level planners to deal with complex real-time problems (Saffiotti, Konolige, & Ruspini 1995). In this way, ABC^2 can be defined as an hybrid model, mainly based in the opportunistic control proposed in (Hayes-Roth *et al.* 1979). Next section describes this model. The formalization of the planning algorithm of ABC^2 is presented in Section and it is compared with traditional nonlinear planners and also with reactive approaches. Then, a discussion about this model is presented, showing its advantages and disadvantages. Some domains where it has been used are presented in order to show its applicability. Finally, Section presents some conclusions.

ABC^2

ABC^2 is a general approach that can be applied to any kind of agent (hardware or software), though the experiments we report all refer to robotic agents. The aim of this model is to allow explicit cooperation among the team members using opportunistic planning to combine agents actions. This actions will be based on predefined skills

The model defines an intelligent autonomous agent as a knowledge structure defined by a set of static and dynamic attributes. Among the static ones, as shown in Figure 1, 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: $A =< N, S, Y, L, H >$

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 in a given moment is defined by $< A_g, I, Q >$.

This model is based on *Skills*, similar concept to classical planning operators, defined as a set of simple and

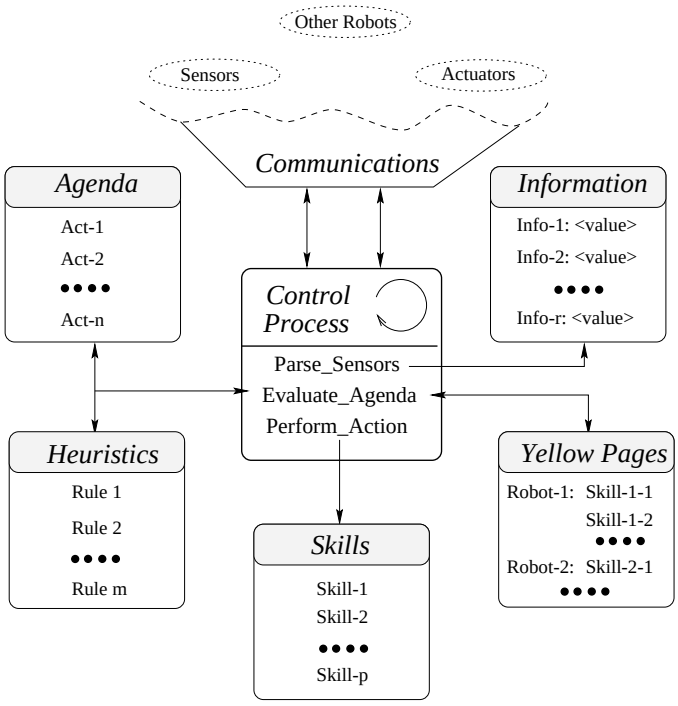

Figure 1: Architecture of the ABC^2 Robots.

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. The definition of a particular skill 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, which is the equivalent of requiring the truth value of the preconditions in classical operators.
- The design and implementation of the controller that performs the desired action represented by a function named *Execute*, 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 of the skill being evaluated but not being able to execute its associated controller (the *Ready* function returns a `FALSE` value). This list has been named *Needs* and simplifies the search of potential operators that provide the preconditions in classical planners.
- Establishing the *Priority* assigned to the behavior. This value will be used by heuristic rules to select acts from an agenda and holds the "a priori" importance of that skill.

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(Itsuki 1995), the inputs are not numerical data obtained from the sen-

sors, 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.

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, named *Communications* in Figure 1 to cope with it.

The *Agenda*, represented in Figure 1, 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 another 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.

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, the number of acts that require an act to be evaluated, information about the environment, 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. The whole algorithm can be summarized to:

Initialize (*Agenda*)
while *Agenda* $\neq \emptyset$
  Recursively *Remove* Act_i from *Agenda* if $Act_i.Called$=0
  $Acts_{app}$= {$Act_i \in$ *Agenda* such that Act_i.Ready=True }
  for all $Act_i s \in$ *Agenda* such that $Act_i s \notin Acts_{app}$ **do**
    if not ($Act_i.Skill$).Expanded **then**
      Expand $Act_i.Skill$
  $Act \Longleftarrow$ Select *Act* from $Acts_{app}$ using $Heuristics$
  Evaluate (*Act*)

where *Agenda* is the agenda of the robot, $Acts_{app}$ the subset of the acts contained in the agenda whose associated skill is *Ready*, and *Initialize* inserts the initial act into the *Agenda*. This loop will finish when the *Agenda* becomes empty.

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 would be required in order to achieve this initial goal, for instance as needs of this act. 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 ($Acts_{app}$) 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 it 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 exists (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.

An Example of use

ABC^2 has been used in different domains. For instance, it has been used two coordinately push an object by to simulated robots (Matellán, Molina, & Sommaruga 1996). This same experiment has also been carried out using real mini-robots (Matellán & Borrajo 1998) with similar results. Another type of environment where it has been successfully used is the competition of simulated robotic soccer players, named RoboCup (Kitano *et al.* 1995). ABC^2 was used directly in this environment, without any adaptation, apart from the design of the specific skills. Each simulated robot was controlled by its own implementation of the model, using its own subset of skills, set of heuristic rules and initial act (Matellán & Borrajo 1997).

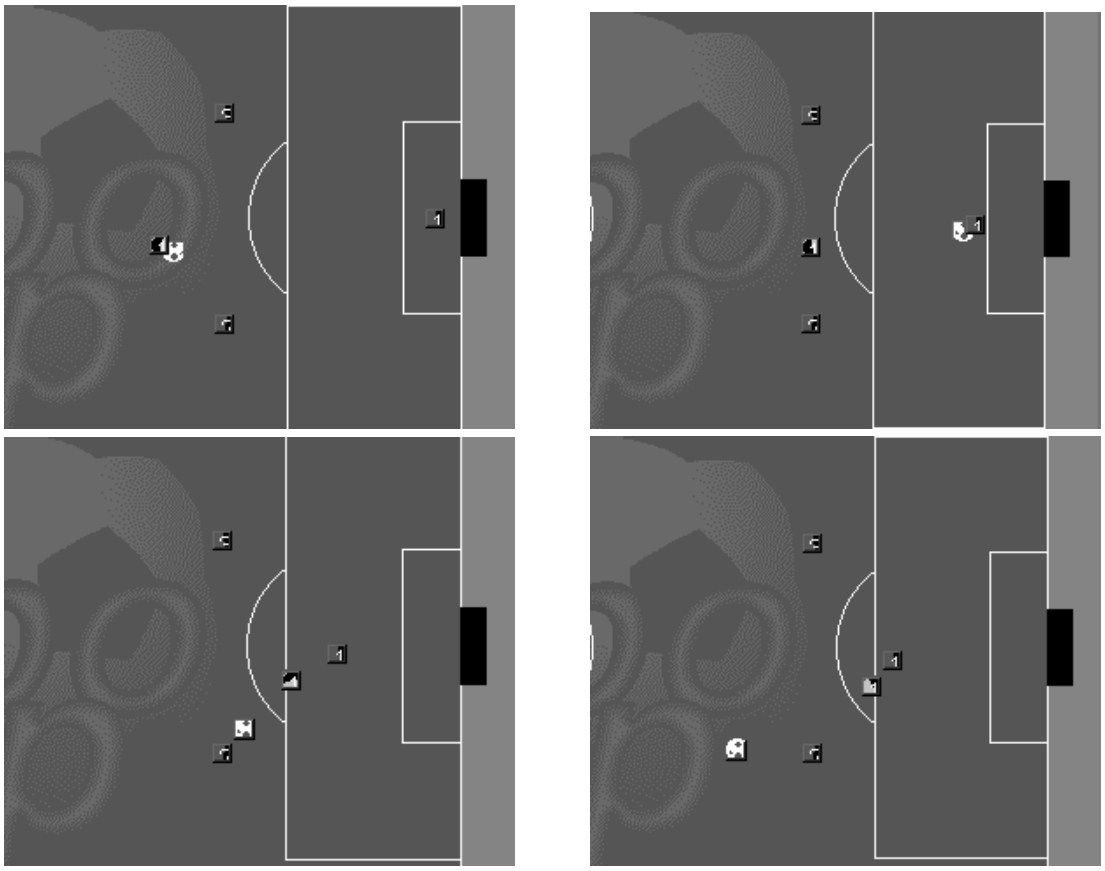

Figure 2: An example in the RoboCup domain.

The example shown in Figure 2 employs a reduced and simplified subset of the skills used in the competition. This example consists of one forwarder (in light gray in Figure 2) that tries to score (using a skill named `Kick_Goal`); a goal-keeper (dark gray player number 1) that will remain in the center of the goal till the ball is near enough to `Came_Out` trying to catch the ball, and, if it succeeds, it will try to `Pass` it to one of the defenders. Defenders will remain in its position waiting for the ball. If the ball come by its surroundings they will kick it towards the opponent goal.

Figure 2 shows the execution of this example. In the first image, the forwarder kicks towards the goal. Then the goal-keeper comes out, and catches the ball. In the third capture, it passes the ball to its team-mate, and in the final one the defendant kicks it towards the opponents goal. An example of the skills used in this experiment is, for instance, the `Kick_Goal` skill. This skill can be defined given the description of its two main features:

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 of the kick.

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.

Once the skills have been implemented, players can be defined. A player consists of 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). In order to show how ADL is used, let us present the configuration of one of the agents of the last example, such as the 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 it 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 goalkeeper
* 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*), 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 current version of ABC^2, fuzzy rules are used to implement the heuristics. These rules are defined in a separate file, in this case in `goalie.heuristics` file. The design of the rules has been made heuristically attending to the experience in previous matches, but learning methods can be used to improve them, because they are defined in a high level language. 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.

ABC^2 model has been tested in different domains, as it has been mentioned in the first paragraph of this section. In those domains, it has shown that it is feasible. That is, it works and it is able to solve usual problems in different domains (Robosoccer, cooperation between robots, cooperative software agents, etc.). The model is also flexible, adaptable to different domains, as

Besides, ABC^2 integrates reactive responses and planning in a convenient way. The agent is able to deal with contingencies in a reactive way, as well as, it can use classical search mechanisms when dealing with high level problems.

ABC^2 vs. classical and reactive systems

Under classical assumptions, planners are 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, into the 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. The execution of the plan obtained is usually trusted to a different module called the scheduler. The scheduler compares the real situation of the world with the one foresight by the planner. If the difference between them is bigger than a given value, a new planning process considering the actual situation is started.

In ABC^2 there is no instantiation of operators, neither the concept of "state", nor any reasoning about foresight states. Actions are decided based on the actual situation of the world and the specific opportunities to act. In this way, it is similar to the reactive approach. However, reactive systems offer only one "optimal" action for execution at each point in time. For example, under the reactive model, a robot triggers and immediately executes the action "avoid obstacle" if it is nearer than a value to a moving obstacle. Runtime events control robot's actions because the reactive model hard-wires the robot's goals hierarchy.

By contrast, ABC^2 uses only enabling conditions for triggering actions and may identify several possible actions at each point in time. For example, the robot in the former example controlled by ABC^2 may not trigger the "avoid obstacle" in some conditions, for instance if its goal in that moment is to stop the moving obstacle. That is, the robot may or may not execute the triggered action depending on its goals, and they can be modified by heuristics. In summary, an agent can change its goals without changing the way its actions are triggered and run-time events enable, but do not control, an agent's actions.

The main difference between ABC^2 and classical planners is the way by which goals are specified. In classical planners they are explicitly defined as world configurations; in ABC^2 they are implicitly described in the definition of the *skills*. This means that the pair ($Operator_i$ - $goal_i$) is indivisible. In the same way, the list of $(Operator_i - goal_i)^*$ that can make executable the ($Operator_i$ (*list of needs*) is also indivisible. When a skill cannot be executed, all its *Needs* are introduced in the agenda. This means that all its *Needs* can be considered as conjunctive preconditions. However, the insertion in the agenda does not mean that all its preconditions have to be executed. Typically, only one of them will be enough to let its parent skill be executable. Thus, skills in the list of needs of another skill should be interpreted as an OR of possibilities.

Another difference is the way planning is carried out. In classical planners, it is based on a search in the tree of possible operators combinations that leads from the initial situation to the desired one or in the plans space. They generate this tree by looking at the effects of available operators, choosing the ones that produce the desired effects, and generating a subgoal with their preconditions. This search can be made because each operator symbolically defines the effects that it produces in the world.

In ABC^2 this search is pre-compiled in the list of

Needs that each skill has. It is not possible to make any search because the *Execute* feature of the skills does not explicitly define its effects. There is no believes to add, or certainties to remove from the world model. It also allows to produce an action that with an uncertain effect. In the same way, the *Ready* function represents the degree of certainty in the fulfillment of its preconditions, where they can be defined over any information available for the agent.

From other point of view, it can be considered that there is some kind of depth-first search. However, the use of the heuristics prevent the agent from being classify in this kind of systems. On the other hand, *Skill* can be implemented using any kind of AI paradigm, which includes searching algorithms.

There are also some similarities and differences in the way pending goals are handled. Classical planners expand the preconditions of the operators considered as new goals. In ABC^2 the *Called* counter guarantees that each goal is defined only once, and also that the more they are required, the more important they become.

In summary, the ABC^2 model resembles traditional planning in some of its components, and it works in some way as reactive systems. This is the main idea: to combine planning and reaction to act in dynamic environments.

Conclusions

In summary, the ABC^2 model resembles traditional planning in some of its components, and it works in some way as reactive systems. Thus, the main idea of the paper is to present a model to combine planning and reaction to act in dynamic environments. So, we have presented a model, named ABC^2 for the control of robots that mixes high speed reaction an decision making with reduced planning capabilities. This model has been tested in different domains showing that not only it can take appropriate decisions, but it can do it in an appropriate time.

This model is well suited for 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.

References

Blythe, J. 1996. Event-based decomposition for reasoning about external changes in planner. In *Conference on Uncertainty in AI*, 27–34.

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

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

Connell, J. H. 1990. *Minimalist Mobile Robotics: A Colony-style Architecture fo a Mobile Robot.* Cambridge, MA: Academic Press. Latas de Coca-Cola.

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

Hayes-Roth, B.; Hayes-Roth, F.; Rosenschein, S.; and Cammarata, S. 1979. Modeling planning as an incremental, opportunistic process. In *Proc. of the 6th IJCAI*, 375–383.

Itsuki, N. 1995. Soccer server: A simulator for RoboCup. In *Proceedings of the AI-Symposium 95: Special Session on RoboCup*, 29–34. Japanese Society for Artficial Intelligence.

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I.; and Osawa, E. 1995. Robocup: The robot world cup initiative. In *Proceedings of the IJCAI-95 Workshop on Entertainment and AI/Life*, 19–24.

Matellán, V., and Borrajo, D. 1997. An agenda based multi-agent architecture. In *Workshop on RoboCup. Fifteenth International Joint Conference on Artificial Intelligence IJCAI-97*, 121–124.

Matellán, V., and Borrajo, D. 1998. ABC^2: An architecture for intelligent autonomous systems. In *Proceedings of the Third IFAC Symposium on Intelligent Autonomous Vehicles*, 57–61.

Matellán, V.; Molina, J. M.; and Sommaruga, L. 1996. Fuzzy multi-agent interaction. In *Proccedings of the IEEE Conference on Systems, Man and Cibernetics*, 1950–1955.

McFarland, D., and Bsser, T. 1993. *Intelligent Behavior in Animals and Robots.* Complex Adaptive Systems. MIT Press.

Neisser, U. 1967. *Cognitive Psicology.* New York: Prentice Hall.

Nilsson, N. J. 1984. Shakey the robot. Technical report, SRI A.I.

Pavlov, I. P. 1927. *Conditioned Reflexes.* Humphrey Milford.

Saffiotti, A.; Konolige, K.; and Ruspini, E. 1995. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence* 76(1-2):481–526.

Walter, G. 1953. *The Living Brain.* Duckworth.

Wiener, N. 1948. *Cybernetics: Control and communication in the animal and the machine.* Wiley.