Concepts and Fuzzy Models for Control and Coordination in Robotics

Andrea Bonarini Matteo Matteucci Marcello Restelli

Politecnico di Milano Arti£cial Intelligence and Robotics Lab
Department of Electronics and Information
Politecnico di Milano, Milan, Italy
Email:{bonarini,matteucc,restelli}@elet.polimi.it

Abstract

In this paper, we propose a unifying modelling paradigm based on fuzzy models to represent concepts on which the coordination and control modules of an autonomous robot operate in a multi-agent environment. It provides a well-founded tool to represent in a compact way the data interpretations needed to reason effectively on what is happening in the world and what is desired to happen. This modelling paradigm makes the design of behavior, planning, and coordination modules easy, since its primitives are simple and expressive. Finally, many aspects of the model can be tuned or learned automatically, thus reducing the development time, providing adaptation to dynamic environments and robustness.

1 Introduction

Robot control architectures have deeply evolved in the last ten years, but there is still considerable debate over the optimal role of internal representation. The most common architectures of autonomous robots integrate the planning activity, which provides goals for the robot, with behavior-based reactivity, which implements simple and fast control modules. In designing this kind of hybrid architectures, most of the issues arise from the connection between the abstract and physical level representations used respectively in the deliberative and reactive components of the system [7]. Although this practice is now common, only few efforts have been done to formalize, unify, and optimize the knowledge representation model in order to seamless integrate the components in the hybrid architecture.

In this paper, we present our approach to knowledge modelling for autonomous robots, aimed at providing a common framework to represent all the knowledge needed by the modules that participate in control and coordination. We de£ne all the conceptual aspects needed to represent this type of knowledge and we introduce fuzzy sets as a tool to support this representation. Fuzzy sets provide a uniform formalism to classify numerical data into conceptual categories, in a robust and ef£cient way. The fuzzy conceptual representation is used by all the modules of our control architecture: *MAP* (*Map Anchors Percepts*) [3] that integrates data from sensors and other data sources building an internal representation of the world, *BRIAN* (*Brian Reacts by Inferential ActioNs*) that manages the behaviors and implements all the reactive functionalities of our system, and *SCARE* (*Scare Coordinates Agents in Robotic Environments*) [4] that coordinates the agent's behaviors and plans their activity,

In this paper, we £rst introduce the knowledge model, implemented in MAP, then we show its application in *BRIAN* and *SCARE* putting in evidence how a common framework can facilitate the integration of different modules. In fact, a uniform knowledge representation makes it possible a coordinated design of the modules, and an ef£cient exchange of information among them. Moreover, this speci£c knowledge modelling approach is particularly suitable for adaptation and learning algorithms, which, given the presence of the same formal constructs at the basis of any module activity, can thus affect any activity in the system.

The next section introduces the conceptual model used in our architecture for integrating the coordination and reactive components, respectively described in section 3 and section 4. This integration is brie¤y analyzed in section 5. After a short review of related works, we discuss some applications in the Robocup and Space Exploration domains.

2 The Conceptual Model

In a robotic environment, agents have to interact with several *physical objects* and this interaction is typically implemented as a perception-action loop. Robots are equipped with sensors perceiving physical characteristics of the environment and they use these *percepts* to build an internal representation of the environment. Once this internal representation is formed, it is possible to use it for deliberative or reactive processing which produces ac-

tions to be executed in the environment.

We focus on the knowledge representation we practically use to face the problem of creating, and maintaining in time, the connection between symbol-level and signal-level representations of the same physical object [7].

We propose to use a two stages process for creating the internal representation of the world. In this process, percepts are used to instantiate a conceptual model of the environment and, once this real-valued representation is obtained, it is modelled by fuzzy predicates to be used in coordination and control.

Percepts are processed by sensing modules (i.e., smart sensors) to produce high level features. Features referring to a speci£c physical object are collected around the same internal representation, referred to as its *perceptual image* and it can be seen as the instance of a *concept*.

In a formal way, a concept C is described by a set of properties de£ned as tuples in the form

$$p \triangleq < label, \mathbb{D}, \rho >, \tag{1}$$

where $label$ denotes the property name, $\mathbb{D}$ is the set of all the possible values for that property given a speci£c representation code (e.g. for the colors we can use either the set $\{red, green, blue, \ldots\}$ or the RGB space $\mathbb{N}^3_{[0,255]}$) and ρ represents a restriction of the domain $\mathbb{D}$ for the property in the speci£c concept.

According to this concepts-based knowledge representation, a property can be either directly perceived, and thus related to a set of high level features coming from the sensors, or it can be derived from other concepts through inference or computation. This approach allows speci£c properties of the concept to provide additional information about a perceptual image, or infer unperceived characteristics.

Depending on the concept and on the speci£c application domain, a property can be classi£ed as *substantial* or *accidental*. Substantial properties are those properties that characterize the immutable part of a concept; for a given object, their values do not change over time, and they can be used for object recognition since they explain the essence of the object they represent. Accidental properties are those properties that do not characterize a concept, their values are speci£c for each conceptual instance, and they can vary over time. Accidental properties cannot be used for object recognition, but are the basis of instance formation, tracking and model validation [3].

As an example of a concept we may consider a ball, described by substantial properties such as shape and color, and accidental properties such as its position.

During the robot activity, data coming from sensors are matched against the concepts in the conceptual model, and, when enough evidence is collected, a concept instance is generated, thus inheriting by default properties eventually not detected by sensors.

Using concepts it is possible to describe both domain speci£c and general knowledge used by an agent during the its activity. To explain how this knowledge is used, we introduce the notion of model $\mathcal{M}$: given D as the set of the known domains, a model $\mathcal{M}_d$ is the set of all the concepts known by the agent referring to the speci£c domain $d \in D$, linked by relationships – structural (e.g., generalization, and specialization) and domain speci£c (e.g., colors and landmark in structured environments). A relationship between concepts can represent:

1. *constraints*: they must be satis£ed by concept instances in order to belong to the model
2. *functions*: they generate property values for a concepts from property values of another
3. *structures*: structural constraint to be used while reasoning about classi£cation and uncertainty

Once the conceptual model $\mathcal{M}$ relative to the established knowledge has been instantiated, we have an internal representation of the environment on which it is possible to evaluate logic predicates, apply inference, or execute behavior control modules. We call this internal representation *domain* and we will denote it with $\mathcal{D}$.

The domain $\mathcal{D}$ is the real-valued base for knowledge processing in the other modules of the system. Concept instances in $\mathcal{D}$ are represented by fuzzy models in the rest of the architecture to face the issue of uncertain and imprecise perception.

Fuzzy predicates are also adopted to represent concept instances related to aspects of the world, goals, and information coming from other agents. They are represented by a *label* λ, its *truth value* μ_λ, computed by fuzzy evaluation of the concept instance properties, and a *reliability value* ξ_λ of the instance. For instance, we may have a predicate represented as

$$< ObstacleInFront, 0.8, 0.9 >$$

which can be expressed as: "It is quite true ($\mu_\lambda = 0.8$, coming from the fuzzy£cation of real-valued properties) that there is an obstacle in front of the robot, and this statement has a reliability quite high ($\xi_\lambda = 0.9$, due to the reliability of the sensing conditions)"

We consider ground and complex fuzzy predicates. *Ground fuzzy predicates* range on concept properties directly available to the agent through $\mathcal{D}$, and have a truth value corresponding to the degree of membership of instance properties to labeled fuzzy sets. The reliability of sensorial data is provided by the anchoring process basing on percepts analysis, and goal reliability is stated by the planner.

A *complex fuzzy predicate* is a composition of fuzzy predicates obtained by fuzzy logic operators. Complex fuzzy predicates organize the basic information contained in ground predicates into a more abstract model. In Robocup, for instance, we can model the concept of ball possession by the *OwnBall* predicate, de£ned by the conjunction of the ground predicates *BallVeryClose* and *BallInFront*, respectively deriving from the fuzzy£cation of the distance and direction properties of the ball concept instance in $\mathcal{D}$.

Some of the most important properties that can be obtained by basing the robot control architecture on the model using the knowledge model are below summarized.

- *noise £ltering*: using a conceptual model of the environment it is possible to eliminate out–layers in percepts and £lter in a proper way noisy data coming from sensors
- *sensor fusion*: percepts coming from different sensors, and referring to the same objects, can be fused thus checking the coherence of perception, enhancing fault tolerance and enabling on–line diagnosis
- *virtual sensing*: a model of the environment can be used to infer new features, not perceived by physical sensors, to be added into the internal representation during sensor fusion;
- *time consistency*: the instances in the conceptual model represent a state of the environment; it is possible to maintain and monitor its consistency in real–time
- *abstraction*: the use of fuzzy predicates instead of raw data, or features, in the behavior de£nition gives more abstraction in designing robot behaviors, and robustness to noisy data; it also facilitates design, since gives the designer the possibility to reason in symbolic terms. Moreover, exchanging this information is more effective for agents of a Multi-Agent System (MAS) sharing the same semantics for symbols.

3 SCARE, the coordination system

Cooperation holds a very important role in multi-agent system applications. To face the typical issues of these applications, we have implemented *SCARE* (*Scare Coordinates Agents in Robotic Environments*) [4] a general architecture for coordination in multi-robot domains. SCARE is able to deal with:

heterogeneity when a MAS is made up of agents with different skills, our architecture exploits these differences in order to improve the overall performance

communication coordination policy may change according to the amount of information that can be exchanged among agents and according to the network connectivity

adaptation in order to grant the autonomy of the system, the coordination mechanism is able to dynamically modify its parameters in reaction to environment changes

Using SCARE, the MAS application developer has to identify the macro-activities that the agents can carry out; we call *jobs* such macro-activities. Besides jobs, we have de£ned other macro-activities named *schemata*. A schema is a complex activity consisting of sequences of jobs that require the collaboration of two or more agents.

The job assignment process is carried out by a special kind of agent called *meta-agent* ($\hat{A}$), through two phases: *decision making* and *coordination*. In the *decision making phase*, $\hat{A}$ computes the suitability of an agent for each activity, through the composition of several parameters, all but the second implemented by fuzzy predicates, which operate on the domain $\mathcal{D}$:

cando de£ne when the activity can take part in the assignment process;

attitude de£ne how much the skills of the agent are useful for the activity;

chance de£ne the situation where the agent has good possibilities to succeed in the activity;

utility de£ne the situation where the activity is useful for the agent team;

success de£ne the goal achievement situation for the activity;

failure de£ne the situation where the activity should be stopped because of unrecoverable failure.

An activity terminates when the success or failure conditions are veri£ed. If an agent is idle, $\hat{A}$ tries to assign it to some activity. $\hat{A}$ £rstly evaluates, for each activity, the cando predicates in order to reject those activities that cannot take place. For each remaining activity, $\hat{A}$ evaluates utility and chance predicates, and the agent's attitude, thus obtaining indicators to take the decision. Through the application of some multi-objective technique (e.g., weighted sums, goal programming, normal-boundary intersection, multilevel programming, or others), the $\hat{A}$ can produce an ordered list of activities (*agenda*), and can start the coordination phase.

The *coordination phase* considers that the agents are not working individually and that they must cooperate to achieve the MAS goals. If we simply assign each agent to

the most suitable activity according to the decision making phase, it may happen that the assigning process does not satisfy some *coordination constraints*, such as: *cardinality* – for each job the designer sets the minimum and maximum cardinality, i.e., the minimum and maximum number of agents that can be assigned to the job at the same time – and *schema coverage*, a schema can be assigned only if there is a suitable agent for each functionality.

In this phase, $\hat{A}$ searches for the best job allocation to agents by observing coordination constraints. How this can be achieved depends on the structure of the MAS communication system and is better discussed in [4]; in that paper we also show some con£gurations which match different qualities and topologies of the communication network.

At the end of this process, each agent (both goal-driven and reactive) is assigned to a job, and new coordination predicates are produced by SCARE for the behavior management system, as enabling conditions, motivations, and virtual perceptions.

4 BRIAN: the behavior management system

In our behavior management system *BRIAN* [2], shown schematically in Figure 1, integration and coordination among behavior modules is achieved using two sets of fuzzy predicates associated to each of them: *CANDO* and *WANT* conditions. In BRIAN we face the issue of controlling the interactions among modules by decoupling them with context conditions described in terms of fuzzy predicates evaluated over internal states, environmental situations present in $\mathcal{D}$, or goals generated by *SCARE*.

CANDO conditions are used to decide if a behavior module is appropriate to the speci£c situation: if they are not veri£ed, the behavior module activation does not make sense. The designer has to put in this set all the fuzzy predicates which have to be true, at least to a signi£cant extent, to give sense to the behavior activation. For instance, in order to consider to kick a ball into the opponent goal, the agent should have the ball control, and it should be oriented towards the goal. This set of conditions has a twofold result: decoupling behaviour design and increasing the computational ef£ciency of the behaviour management system.

WANT conditions are predicates that represent the motivation for an agent to execute a behavior with respect to the actual context. They may come either from the environmental context (e.g., *BallInFront*, i.e. "the ball is in front of me"), or from strategic goals (e.g., *CollectDocuments*, "I have to collect the documents to be delivered"). Composition of behaviors modules active at the same time is implemented by these WANT conditions representing the opportunity of executing the action proposed by each behavioral module in the speci£c context.

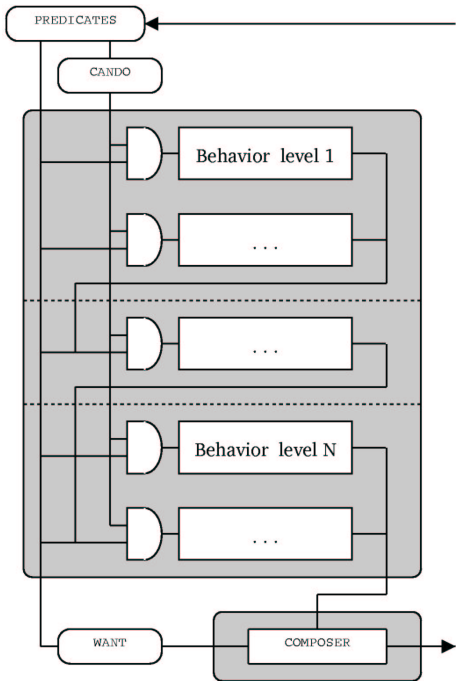

Figure 1: *The behavior management system*

The use of these two different sets of conditions allows the designer to design a dynamic network of behavior modules de£ned by means of context predicates. This is the main improvement with respect to usual behavior-based architectures: we do not have a complex prede£ned interaction schema that has to take into account all possible execution contexts. At any time, the agent knows that it could execute only a restricted set of behavior modules (i.e., those enabled by the CANDO conditions), and it has to select/merge them according to its present motivations.

In BRIAN's implementation, each behavioral module receives data in input and provides output commands to be issued to the environment. We do not make any hypothesis about the implementation of a behavioral module, it can be considered as a mapping from input variables to output variables:

$$B_i : I_i \mapsto A_i \quad I_i \subseteq I = \{i_j\},\ A_i \subseteq A = \{\vec{a}_k\}, \quad (2)$$

where I_i is the speci£c set of input variables for module i, and A_i is the set of its actions.

Since tasks have typically different priorities, it may happen that, in particular situations, some behavioral modules would like to £lter the actions proposed by less critical modules. Let us consider, for example, the case of the *AvoidObstacles* behavior. The related behavior module contains some rules which become active when it is detected an obstacle on the path followed by the robot, and they produce actions which modify the current trajectory in order to avoid collisions. The problem that

we must face with the simple model described by Equation 2 is that the actions proposed by the *AvoidObstacles* module are composed with the output of the other behavioral modules, thus producing unexpected and undesirable results. One possible solution consists in disabling any other behavior module while the *AvoidObstacles* is active. This approach achieves the aim of avoiding collisions with other objects, but it has some drawbacks. If the *AvoidObstacles* module is the only active behavior module, it avoids the obstacles by taking a trajectory that is independent from the one that would have been produced by the other behavior modules, thus degrading the performances.

To overcome the above described problem, we have adopted the hierarchical organization of the behavior modules depicted in Figure 1. We associate a level number to each behavioral module. Any module at level l receives as input, besides the sensorial data, also the actions proposed by the modules which belong to the level $l-1$. Behaviors (except those at the £rst level) can predicate on the actions proposed by modules at lower levels, inhibit some actions, and propose other actions. Let us rewrite Equation 2 for the i-th behavior of level l:

$$B_i^l : I_i \times A^{l-1} \mapsto A_i^l \cup \bar{A}_i^l, \tag{3}$$

where A^{l-1} is the set of the actions proposed by all the modules at level $l-1$, A_i^l is the set of the action proposed by the behavior, while $\bar{A}_i^l$ is the set of actions that should be inhibited.

With this organization, behavioral modules with higher priority must be placed in the higher levels of the hierarchy, so that they can modify the proposed actions in order to manage unforeseen situations. For example, if we place the *AvoidObstacle* module at the second level, we could write rules of this kind: "*if* I am facing any obstacle that is near *and* I would like to move forward *then* cancel all the actions that propose a forward movement". Whenever the robot has no obstacle in its proximity, the *AvoidObstacle* module passes all the proposed actions unchanged to the following level.

Apparently this design approach is in contrast with the behavior-independency principle, but actually it is the opposite. Consider the *KeepBall* behavior, taken again from the RoboCup domain. The goal of this behavior is to hold the ball inside the kicker while the robot is moving. If we think that, when the robot takes the ball, it is of primary importance to maintain the ball possession, we must put the *KeepBall* module at the highest level, so that it can, if necessary, modify the proposed actions as much as it is needed for keeping the ball inside the kicker. In a single level architecture, the most effective way of implementing this *KeepBall* behavior is to embedded it in each behavioral module, thus complicating the rules and really breaking the principle of modularity.

5 Embedding plans in reactive control

Several robotic control architectures present both planning and reactive modules. The main difference between these approaches is the interaction of the two modules. In many approaches [1] [9], the planning module simply activates and deactivates behaviors in order to accomplish more abstract goals. In our architecture, the execution of plans is completely integrated in the reactive behavior engine. This integration does not require any change to the structure of BRIAN, since it operates using the same kind of fuzzy predicates.

We de£ne *parametric* any behavior that, in its input space, has at least one predicate on computed data , and we call *purely reactive* those behavioral modules that predicate only on perceived data.

For instance, we can consider the behavior *GoToTarget*, whose goal is to move the robot in a certain position. The related behavioral module is parametric since the data that identify the target are, in general, not simply perceptions acquired by sensors. Although this behavior module is very simple in its implementation, it can be used by the planning layer to implement very complex activities, that would be quite hard (or even impossible) to achieve with only purely reactive behaviors.

This approach in design allows to de£ne only a few basic parametric behavioral modules, that, opportunely combined, can implement complex plans which are embedded in a reactive context. These modules are typically placed at the £rst levels of the behavior hierarchy, since a plan can be executed as long as the situation on which the plan was generated holds. In environments where situations quickly change, it may often happen that plans fail, thus requiring a re-planning activity that can be computationally expensive and ineffective since only a small part of the plan is exploited. Our approach reduce the re-planning frequency through the introduction of pure reactive behavioral modules in the higher levels of the hierarchy. When the environment evolves to unpredicted situations, some of these modules become active and reactively modify the plan execution. Since, according to our behavior architecture, each behavioral module knows the output of the modules at lower levels, pure reactive modules know what the plan would do, and change it as much as needed to avoid undesired consequences.

SCARE interacts with BRIAN sending to it two kinds of predicates: *coordination predicates*, which in¤uence the enabling conditions, and *perceptual predicates*, which are used by the parametric behavioral modules. coordination predicates refer simply to the job that is assigned to the agent. This information allows to activate only those behavior modules that are necessary to the execution of the assigned job. Perceptual predicates are all the information that BRIAN needs to execute the job, and it is not directly available through the percepts. At each iteration,

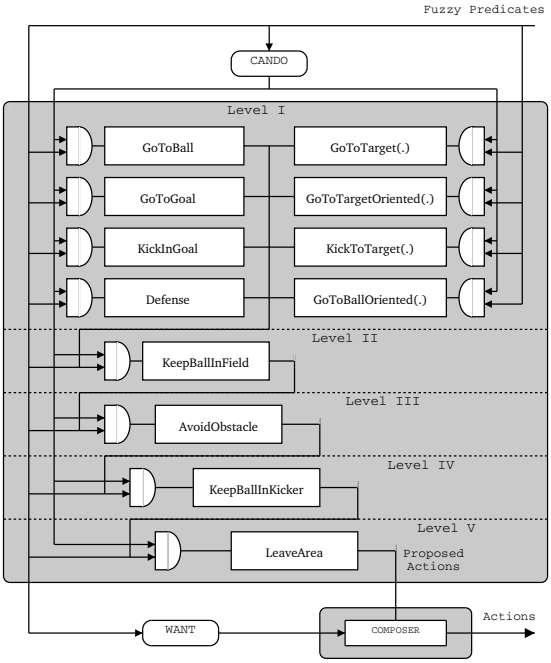

Figure 2: *The behavior con£guration used in RoboCup*

SCARE sends both the coordination and perceptual predicates, and it monitors the state of execution of the assigned job. When a termination condition (both success and failure) occurs, SCARE starts a new job assignment process, in order to identify the best activity, considering the multi-agent context.

6 Experimental results

We have successfully employed our architecture in several robotic applications, where we have veri£ed the versatility of the proposed approach. Both in mono and multi–agent systems, we exploited the bene£ts deriving from the interaction between the planning and the reactive modules. The use of a conceptual model allows an ease and quick development of the high–level control structures. In the following, in order to show the effectiveness of our method, we describe two case studies: *RoboCup* and *Roby*.

6.1 RoboCup

RoboCup is a multi-robot domain with both cooperative and adversarial aspects, suited to test on the effectiveness of our approach. In the coordination layer we have de£ned several jobs (*RecoverWanderingBall*, *BringBall*, *Defense*, etc.) and schemata (*DefensiveDoubling*, *PassageToMate*, *Blocking*, etc.). These activities are executed through the interactions of several behavioral modules (see Figure 2). We have organized our modules in a £ve levels hierarchy. At the £rst level we have put those behavioral modules whose activation is determined also by the information coming from the planning layer. At this level we £nd both purely reactive modules and parametric modules (the latter can be distinguished by a couple of round brackets after their name). The higher levels contain only purely reactive behavioral modules, whose aim is to manage critical situations (such as avoiding collisions, or leaving the area after ten seconds).

To give an idea on how SCARE interacts with BRIAN in this domain, let us consider the behavior *MarkOpponent*. The goal of this behavior is to obstruct the way between the ball and a certain opponent robot. Once SCARE decides that our robot must mark an opponent, there are several ways in which this can be achieved with BRIAN. We can de£ne the parametric behavioral module *GoBetween*, that takes as input the polar coordinates of two points and its goal is to put the robot in a position along the joining line. SCARE activates this behavior which receives as input the polar coordinates of the ball and of the opponent. Using another approach, SCARE can compute the polar coordinates of a point on the joining line and then activate the *GoToTarget* module. A further approach, consists in employing a planning algorithm in order to £nd a free trajectory that leads on a point belonging to the line that joins the ball with the opponent. The trajectory will be followed by the robot by the activation of the *GoToTarget* module with different values of its parameter. These three modalities are ordered by increasing complexity in SCARE, and this is related to the easier implementation of the behavioral modules. In fact, the implementation of the *GoBetween* module is more dif£cult than the implementation of the *GoToTarget* module. In the third approach it is not even necessary the interaction with the *AvoidObstacle* module, since it is all managed at the planning level. On the other hand, in environments where situations change very quickly, it would be necessary to often re-plan everything. We have adopted the second approach because, by exploiting the potentiality of both SCARE and BRIAN, it keeps low the complexity of the whole system.

Let us now consider the *ToyTrain* schema. This activity must be carried out by two robots that go towards the adversarial goal bringing the ball between their bodies. We call *leader* the robot that drives the toy train, while the *follower* is the robot that follows the leader. When the schema is instantiated the two robots must reach their starting positions through the activation of the *GoToBallOriented* parametric module. Once both of them have reached their positions, the leader activates the *GoToGoal* behavior, which takes the robot towards the adversarial goal, while the follower activates two modules: *GoToTarget* and *KeepBall*. The former is a parametric behavioral module that receives as parameter the polar coordinates of the leader, while the former is a purely reactive module that takes care of holding the ball inside the kicker. The interaction of these behaviors (to which we must add

Figure 3: *Roby avoiding an unforeseen obstacle on its path*

the *AvoidObstacle* that is always active) allows the robots to move in the direction chosen by the leader while holding the ball between them. When the toy train is quite near to the goal, the leader moves towards a side of the £eld using the *GoToTarget* module, so that the follower can try to score through the activation of the *KickInGoal* module.

6.2 Roby

Roby is a project (in the framework of the European project GALILEO) that we have developed in collaboration with an Italian company (InfoSolution). In this project, a robot, equipped with a GPS device, must travel between two points, chosen over a pre–compiled map of an outdoor environment. Given the map of the environment and the two ends of the path, the planning module computes a sequence of points which describe a free path. Since the map contains only information about static objects (such as buildings, bridges, etc.), the robot is also equipped with a sonar belt in order to detect obstacles, so that it can handle situations in which the trajectory produced by the planning method is obstructed by something like cars, trees, people, and so on. At each path point the planner associates also a velocity vector that identi£es the orientation and the speed with which the robot should pass over the path point.

Even if this is a mono–robot domain, the coordination level, implemented in SCARE, has been used in order to allow BRIAN to follow the sequence of path-points. We have de£ned the schema *FollowPath* that consists in a sequence of jobs *ReachPathPoint*, one for each vertex of the trajectory. When the robot is assigned to the i-th *ReachPathPoint* job, SCARE calculates the polar coordinates of the path point and the difference between the actual values of orientation and speed and those associated to the current path point, and sends them to BRIAN. The behavior hierarchy implemented for this application has three levels. At the £rst two levels there are two parametric behavioral modules: *GoToTarget* and *ApproachTarget*. The £rst module takes the robot towards the speci£ed path point, while the second one aims to respect the orientation and speed constraints. At the third level we have put the purely reactive *AvoidObstacle* module, since we do not want that our robot suffers any damage caused by collisions. During the navigation the data collected from the sonar belt, besides to be used by the *AvoidObstacle* module, are used to integrate the information stored in the environmental map. The i-th *ReachPathPoint* job terminates when either the robot reaches the related path point (success condition), or a timeout expires (failure condition). In the former case, the current job ends, and the schema activates the *ReachPathPoint* job related to the (i+1)-th path point. When the last path point is reached the *FollowPath* £nishes with success. In the latter case, the whole schema is aborted, and the planning module is reactivated in order to produce, starting from the current robot position, a new sequence of path points. Although the implemented structure is very simple, we have obtained satisfactory results. We have made several trials with different conditions, and the robot was always able to reach the goal point, and seldom was required the replanning activity.

7 Related works

To overcome the issues of both the deliberative control approach and the reactive control approach, many researchers moved towards hybrid systems looking for the appropriate balance between reactivity and deliberation. These architectures can be roughly classi£ed in hierarchical and non-hierarchical systems.

Simmons' Task Control Architecture (TCA) [13] does not have tiers. TCA represents each task through a task net where each node can be decomposed further or is a primitive which interfaces with the robot, or other nodes, through messages. TCA tasks trees are not explicitly connected, and are manipulated directly by C function calls. The lack of a representation for task trees makes very dif£cult the use of a general planner, since its output should be translated into C code and compiled.

In DAMN [11] a set of behaviors votes for or against the possible set of actions of the agent. Each behavior gives a weight that is established by the mode manager according to the given context. Then, a voter selects the action that is associated with the highest weighted sum of the received votes. DAMN follows a non-hierarchical approach, where the deliberative components (like other system components) declare their preferences by voting the set of possible actions.

In hierarchical hybrid architectures, a basic reactive layer is controlled by a planner or sequencer. AURA [1] is one of the £rst hybrid multi-layer architectures, and, like many others, it is speci£c to mobile robot navigation. At the bottom level there are motor schemes, i.e., a vector £eld associated with a goal or an obstacle. A second level enables to combine these motor schemes in order to pro-

duce the desired effect. At this second level, AURA generates a world model that is connected directly to the low level motor schemes.

Particular success gained the three-layer architectures. These control architectures consist of three main components: a reactive control mechanism, a deliberative planner, and a sequencing mechanism that connects the other two components. Connell, on the basis of the Subsumption architecture, developed SSS [6]. At the top level of SSS there is a symbolic layer, where, through complex data structures, goals are represented. The ability of SSS to respond to contingencies relies on the Subsumption architecture adopted in the middle layer. At the bottom layer there is a collection of traditional servo-control loops. Other three-layer architectures are ATLANTIS [8] and 3T [5], which grew out of the same work. At the top level there is the deliberative tier, where a planner synthesizes all the goals into a partially-ordered plan listing tasks for the robot to perform. Each task corresponds to one or more sets of sequenced actions, that are stored in the sequencing tier. At this level, the selected action is decomposed into other sequenced actions and £nally activates a speci£c set of skills contained in the reactive layer. The main difference between ATLANTIS and 3T is that ATLANTIS leaves more control to the sequencing tier, which speci£cally calls the deliberative tier. Noreils [10] proposed a three levels (called planning, control, and functional) architecture similar to those described above. The principal difference is in the middle level, where Noreils' formalism distinguishes between failures and non-failures.

Other researchers have adopted fuzzy models to implement control systems in autonomous robots [12], since it has been demonstrated that they constitute a powerful and robust modeling paradigm close to the designers knowledge models.

8 Conclusion

In this paper we have presented the fuzzy cognitive model we use to integrate in a uniform framework the deliberative and reactive components of our multi-agent systems. The cognitive model we propose, integrates coordination, planning and reactive behaviors providing a common cognitive substratum for a team of robots, where behavioral modules are used as high-level macro-actions that a ¤exible multi-agent coordination system uses to compose structured plans.

From the multi-agent system perspective, the use of such unifying cognitive model provides an effective tool for seamless integration of the heterogeneous members in the team, gaining as much as possible from the presence of different skills and competencies in the robots. Moreover, all the elements in the architecture are based on simple fuzzy predicates that can easily be managed, designed and adapted. Doing it this way, the control model can be easily designed to be tuned and adapted on-line so that the team strategies and the role of robots in the control schemata can be automatically modi£ed to face different opponent teams, and changes in robot performances.

References

[1] R. C. Arkin. Towards the uni£cation of navigational planning and reactive control. In *Proc. of the AAAI Spring Symposium on Robot Navigation*, pages 1–5, 1989.

[2] A. Bonarini, G. Invernizzi, T.H. Labella, and M. Matteucci. An architecture to co-ordinate fuzzy behaviors to control an autonomous robot. *Fuzzy Sets and Systems*, 134(1):101–115, 2002.

[3] A. Bonarini, M. Matteucci, and M. Restelli. Anchoring: do we need new solutions to an old problem or do we have old solutions for a new problem? In *Proceedings of the AAAI Fall Symposium on Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems*, page In press, Menlo Park, CA, 2001. AAAI Press.

[4] A. Bonarini and M. Restelli. An architecture to implement agents co-operating in dynamic environments. In *Proc. of AAMAS 2002 - Autonomous Agents and Multi-Agent Systems*, pages 1143–1144, New York, NY, 2002. ACM Press.

[5] R. Peter Bonasso, James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Marc G. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Arti£cial Intelligence*, 9(2/3):237–256, April 1997.

[6] Jonathon H. Connell. Sss: A hybrid architecture applied to robot navigation. In *Proceedings IEEE International Conference on Robotics and Automation*, 1992.

[7] S. Coradeschi and A. Saf£otti. Anchoring symbols to sensor data: preliminary report. In *Proceedings of the 17th AAAI Conf*, pages 129–135, Austin, Texas, 2000.

[8] Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the National Conference on Arti£cial Intelligence (AAAI)*, 1992.

[9] K. Konolige, K.L. Myers, E.H. Ruspini, and A. Saf£otti. The Saphira architecture: A design for autonomy. *Journal of experimental & theoretical arti£cial intelligence: JETAI*, 9(1):215–235, 1997.

[10] Fabric Noreils and Raja Chatila. Plan execution monitoring and control architecture for mobile robots. *IEEE Transactions on Robotics and Automation*, 2, 1995.

[11] Julio K. Rosenblatt. Damn: A distributed architecture for mobile robot navigation. In *Proceedings of the AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*, Stanford, CA, 1995. AAAI Press.

[12] A. Saf£otti, K. Konolige, and E. H. Ruspini. A multivalued-logic approach to integrating planning and control. *Arti£cial Intelligence Journal*, 76(1-2):481–526, 1995.

[13] Reid Simmons. An architecture for coordinating planning, sensing and acting. In *Proc. of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, 1990.