Filling the Gap among Coordination, Planning, and Reaction Using a Fuzzy Cognitive Model

Andrea Bonarini, Matteo Matteucci, and Marcello Restelli

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

Abstract. Coordination, planning, and reactivity are important for successful teams of autonomous robots, in dynamic adversarial domains. In this paper, we propose a fuzzy cognitive model to integrate coordination, planning and reactive behaviors in a team of cooperating robots. In our architecture, behavioral modules are used as high-level macro-actions that compose structured plans defined by a flexible multi-agent coordination system. The use of an unifying cognitive model provides an effective tool for seamless integration of reactive and deliberative components in the robots, gaining as much as possible from the presence of different skills and competencies in the team. The control model is 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 modified to face different opponent teams, and changes in robot capabilities.

1 Introduction

Since some years, the architecture of autonomous robots integrates the traditional planning activity, which provides goals for the robot, with behavior-based architecture that 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 [6].

Usual solutions propose an ad-hoc integration, without any high-level cognitive model as a uniform substratum for the overall system. The architecture we are proposing in this paper is aimed at integrating coordination, planning, and reactivity using such a cognitive approach where agents have internal models, intentions, goals, and can still react to unexpected events. This cognitive model is implemented by *fuzzy predicates* that we use to represent *concepts*.

In our architecture, the perception-action loop is performed through an abstraction process that, starting from raw data gathered from sensors, produces the high-level concepts used in planning, coordination, and execution. On the first stage of this process, we have the *Sensing* activity: intelligent sensors (i.e., sensors with some computational capabilities) process the raw data stream into

D. Polani et al. (Eds.): RoboCup 2003, LNAI 3020, pp. 662–669, 2004.

features to be used on the next stage for *Data Interpretation*. For instance, omnidirectional vision systems [2] provide relative positions and distance of colored objects in the Robocup domain.

Data interpretation is obtained using *MAP* (*MAP Anchors Percepts*) [4] to realize sensor fusion, anchoring, and self-localization. MAP is suitable for enhancing the local internal model of each agent in a Multi-Agent System (MAS) also with information coming from other agents obtaining, in such a way, global distributed sensing, and map integration. The results of this data interpretation is a real-valued internal model on which are evaluated basic fuzzy predicates that describe the percepts of the robot in the higher level formalism used for knowledge processing both in deliberative and reactive components.

The use of fuzzy predicates gives the possibility to represent both interpretation of data coming from sensors, and high level abstractions coming from a deliberative process involving planning, coordination and communication. The selection of a fuzzy representation makes it possible to face, with an intrinsically robust model, uncertainty and approximation, unavoidable in applications interfaced with the real world. At the same time, fuzzy predicates can be quite effectively automatically adapted to changing situations. Our robots represent by fuzzy predicates also their intentions and goals, which are in this way integrated with data in a unique, simple, adaptive model.

Coordination among robots is defined using the deliberative component described in the next section. In this activity, we consider also aggregated information about the skills of each robot and their appropriateness in a given situation. These are parameters that can be easily defined and tuned on-line in a relatively short time, and can affect the behavior of the single robot and of the whole team. This results in a team able to adapt on-line to different types of opponents, and to possible performance degradation of its members. On the reactive side of the control loop, we have a behaviors management system that maps fuzzy predicates into actions to be executed by the robot actuators. Actions are weighted considering their applicability, intentions and goals, so that the behavior of a robot depends both on actions proposed by reactive module, desires, and intentions.

In the next section we give a brief description of the deliberative component in charge of coordination and long-term planning in our architecture. Section 3 introduces the behavior management system used in our robotics applications, and in the section that follows we describe how deliberative and reactive component can be easily integrated by mean of the underlying cognitive model that we have previously introduced. Section 5 presents a robotic application in which we have adopted our cognitive approach as a case study.

2 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*) [5] 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: *jobs* and *schemata*. A schema is a complex activity consisting of sequences of jobs that require the collaboration of two or more agents. A job is defined by the goals that should be achieved, and each agent is free to select the procedures to achieve these goals according to its own capabilities.

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:

cando define when the activity can take part in the assignment process;

attitude define how much the skills of an agent are useful for the activity;

chance define the situation where the agent has good possibilities to succeed in the activity;

utility define the situation where the activity is useful for the agent team;

success define the goal achievement situation for the activity;

failure define the situation where the activity should be stopped because of unrecoverable failure.

An activity terminates when the success or failure condition is verified. If an agent is idle, $\hat{A}$ tries to assign it to some activity. $\hat{A}$ firstly 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. $\hat{A}$ obtains an ordered list of activities (*agenda*) by solving the optimization problem, 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 constraint*, 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 [5]; in that paper we also show some configurations which match different qualities and topologies of the communication network.

At the end of this process, each agent is assigned to a job, and new fuzzy predicates are produced by SCARE to be used in the behavior management system.

3 BRIAN: The Behavior Management System

Especially in dynamic environments, the main approach to robot control design is the so called behavior-based architecture, where the robot is controlled by the implicit cooperative activity of behavioral modules. Each module operates on a small subset of the input space implementing a relatively simple mapping from sensorial input to actions.

In our behavior management system *BRIAN* (*Brian Reactively Implements AgeNts*) [3], we face the issue of controlling the interactions among modules by decoupling them with context conditions described in terms of internal state, environmental situation, goals, and communications with other agents. All these concepts are defined using the cognitive model introduced in Section 1, based on fuzzy predicates. Integration and coordination among behavior modules is achieved using two sets of fuzzy predicates associated to each of them: *CANDO* and *WANT* conditions. CANDO conditions are used to decide whether a behavior module is appropriate to the specific situation. 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.

WANT conditions represent the motivation for an agent to execute a behavior in the present situation. They may come either from the environmental context, or from strategic goals.

The use of these two different sets of conditions allows the designer to design a dynamic network of behavior modules defined by means of context predicates. This is our main improvement with respect to usual behavior-based architectures; we do not have any complex, predefined interaction schema that has to take into account all possible execution contexts.

Since tasks have typically different priorities, it may happen that, in particular situations, some behavioral modules would like to filter the actions proposed by less critical modules. Let us consider, for example, the case of the *AvoidObstacle* behavior; it is implemented by rules which become active when an obstacle is detected 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 is that the actions proposed by the *AvoidObstacle* module are composed with the output of the other behavioral modules, thus possibly producing unexpected and undesirable results. One possible solution consists in disabling any other behavior module while the *AvoidObstacle* is active. This approach

achieves the aim of avoiding collisions with other objects, but it has some drawbacks. If the *AvoidObstacle* 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. Moreover, if we have several levels of priority, with several behavioral modules for each level, this approach turns out to be very awkward.

To overcome the previously described problem, we have adopted a hierarchical organization of the behavior modules 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 first level) can predicate on the actions proposed by modules at lower levels, inhibit some actions, and propose other actions. For the i-th behavior of level l we have:

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

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 $\overline{A}_i^l$ 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 urgencies. 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 may seem in contrast with the behavior-independency principle, but it is actually 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 embed it in each behavioral module, thus complicating the rules and really breaking the principle of modularity. In our architecture, dependencies among modules are explicit in the hierarchical structure, and the interface is limited to the action proposed by the modules.

4 Embedding Plans in Reactive Control

Several robotic control architectures present both planning and reactive modules. The main difference between these approaches is the interaction between the two modules. In many approaches [1] [7], the planning module simply activates and deactivates behaviors in order to achieve more abstract goals. In our architecture, the execution of plans is completely integrated in the reactive behavior engine.

SCARE interacts with BRIAN sending to it two kinds of predicates: *Coordination predicates*, which influence 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 world model. At each iteration, SCARE sends both the Coordination and Perceptual predicates, and it monitors the state of execution of the assigned job. When a termination condition (either success or failure) occurs, SCARE starts a new job assignment process, in order to identify the best activity, considering the multi-agent context.

5 Experimental Results

We have successfully employed our architecture in several robotic applications, where we have verified the versatility of the proposed approach. Both in mono and multi-agent systems, we exploited the benefits deriving from the interaction between the coordination, planning, and reactive modules. The use of a conceptual model allows an easy and quick development of the high-level control structures. In the following, in order to show the effectiveness of our method, we describe a case study in the RoboCup context.

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 defined 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 1). We have organized our modules in a five levels hierarchy. At the lower level we have put the behavioral modules whose activation is determined also by the information coming from the planning layer. At this level we find 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). The activation of these modules does not depend on the high level activity. In order to give an idea of how the whole system works, we present a couple of examples.

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 by BRIAN. We could define the parametric behavioral module *GoBetween*, that takes as input the polar coordinates of two points and has the goal 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 could compute the polar coordinates of a point on the joining line and then activate the *GoToTarget* module.

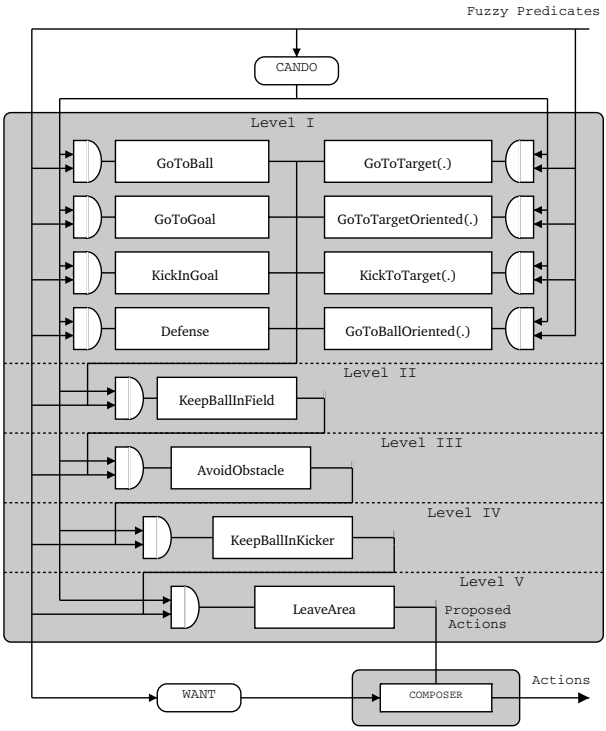

Fig. 1. The behavior configuration used in RoboCup

A further approach, consists in employing a planning algorithm in order to find 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 through the activation of the *GoToTarget* module with different values of its parameter. These three modalities are ordered by increasing complexity from the SCARE point of view, that implies a simplification for what concerns the behavior modules. In fact, the implementation of the *GoBetween* module is more difficult 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 always replan 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.

6 Conclusion

Robot control architectures have deeply evolved in the last twenty years, but there is still considerable debate over the optimal role of internal representation.

In this paper we have presented the fuzzy cognitive model we use to integrate in a uniform framework the deliberative and reactive components of multi-agents

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 compose structured plans defined by a flexible multi-agent coordination system.

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 knowledge processing level 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 modified to face different opponent teams, and changes in robot performances.

References

1. R.C. Arkin. Towards the unification of navigational planning and reactive control. In *Proc. of the AAAI Spring Symposium on Robot Navigation*, pages 1–5, 1989.
2. A. Bonarini. The body, the mind or the eye, first? In M. Asada M. Veloso, E. Pagello, editor, *RoboCup 98–Robot Soccer World Cup III*, pages 695–698, Berlin, D, 2000. Springer Verlag.
3. A. Bonarini, G. Invernizzi, T.H. Labella, and M. Matteucci. An architecture to coordinate fuzzy behaviors to control an autonomous robot. *Fuzzy Sets and Systems*, 134(1):101–115, 2002.
4. A. Bonarini, M. Matteucci, and M. Restelli. A framework for robust sensing in multi-agent systems. In *RoboCup 2001 - Robot Soccer World Cup V*, Berlin, D, 2001. Springer Verlag.
5. A. Bonarini and M. Restelli. An architecture to implement agents co-operating in dynamic environments. In *Proceedings of AAMAS 2002 - Autonomous Agents and Multi-Agent Systems*, pages 1143–1144, New York, NY, 2002. ACM Press.
6. S. Coradeschi and A. Saffiotti. Anchoring symbols to sensor data: preliminary report. In *Proceedings of the 17th AAAI Conf*, pages 129–135, Austin, Texas, 2000.
7. K. Konolige, K.L. Myers, E.H. Ruspini, and A. Saffiotti. The Saphira architecture: A design for autonomy. *Journal of experimental & theoretical artificial intelligence: JETAI*, 9(1):215–235, 1997.