From a Concurrent Architecture to a Concurrent Autonomous Agents Architecture

Augusto Loureiro da Costa
loureiro@lcmi.ufsc.br

Guilherme Bittencourt
gb@lcmi.ufsc.br

Department of Automation and Systems
Federal University of Santa Catarina
CEP 88.040-900 - Brazil

Abstract. In this paper, the autonomous agent architecture used to implement the RoboCup simulator league UFSC-Team is presented. This architecture consists of three concurrent processes that encapsulate different inference engines. These take decisions in three different levels, called reactive, instinctive and cognitive. This architecture is an evolution of the concurrent architecture for cognitive multi-agents, used in the implementation of the UFSC-Team'98 that has participated in the RoboCup'98. The present implementation was designed to solve some agent synchronization and real-time response problems presented by the old architecture, due mainly to its centralized decision approach.

1 Introduction

In its first participation in the simulator league of the RoboCup'98, the UFSC-Team presented a concurrent cognitive multi-agent architecture [12]. The idea was to implement perception, action, communication, cooperation, planning and decision making exploring the concurrent programming approach [1].

The first concurrent architecture was based on three processes: interface, coordinator and expert. The interface was designed to handle perception and action. The agent/environment interaction supported by the Soccerserver consists of message exchange using a Inet Domain Socket channel. The perception information is received and the action commands are sent through this same channel. The function of the process interface was just to translate the perception and communication information into the Parla language [10] (the Agent Communication Language used by the UFSC-Team agents) and expressions from the Parla language to Soccerserver commands.

The process coordinator was responsible for the agent communication and for starting and conducting the cooperation processes. According to the original architecture proposed in the Expert-Coop environment [9], this process was responsible for the inter-process communication management, i.e., it should receive directly the messages sent by other agents and handle them. But, according to the RoboCup simulator league rules, all inter-agent communication must be done only through the Soccerserver. Because of this, the inter-process communication and the perception information are all received through the same Inet

M. Veloso, E. Pagello, and H. Kitano (Eds.): RoboCup-99, LNAI 1856, pp. 274–285, 2000.
Springer-Verlag Berlin Heidelberg 2000

Domain socket channel. Therefore, in this implementation, the process interface also received the inter-agent messages and forwarded them to be handled by the process coordinator, along with the perception information and referee messages.

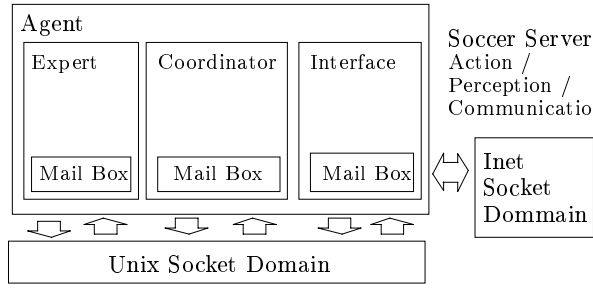

Fig. 1. The Concurrent Architecture

Finally the process expert was responsible by planing and decision making. It had a knowledge-based system encapsulated where the perception information, the messages from the referee and from other UFSC-Team agents were stored and used to infer appropriate decisions, according to the knowledge-based system rules. These three processes communicated among them by message exchange using sockets into the Unix domain (see Figure 1).

This first concurrent implementation, with a centralized decision approach, has presented some problems with agent/environment synchronization and the response time was considered too high. In fact, the best real-time responses presented by the UFSC-Team'98 agent architecture were between 70 and 80 ms, even using Case-Based Reasoning [2] to split the knowledge into different packets. Beside this, the knowledge-based system responsible for the agent decision making became very complex, because it had to include rules to treat information from high level, like what kind of collective play should be chosen in a given situation or what agents can be joined into a known play, to low level, like which power dash value or which turn value should be chosen.

To solve these problems, the agent architecture used in UFSC-Team has migrated from a concurrent approach with centralized decision making, to an autonomous agents architecture, inspired by the architecture proposed in [4], with three decision levels – reactive, instinctive and cognitive – implemented in a concurrent way (see Figure 2). The concurrent model was kept with the same three processes: interface, coordinator and expert. But now each one of these processes encapsulates a different inference engine and is responsible for one of the three decision levels. Both the first implementation and the current one were written in the C++ programming language and they integrate a partial implementation of the environment to develop cognitive multi-agent systems under real-time restrictions called Expert-Coop++.

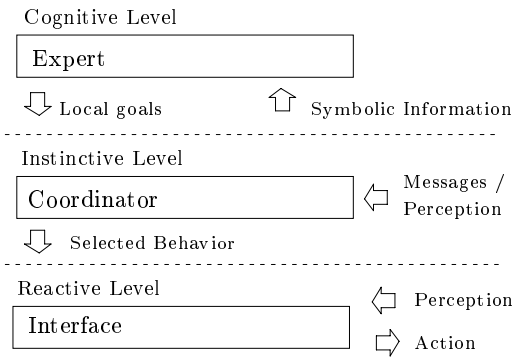

Fig. 2. The agent information flow

The reactive level inference engine is implemented in the interface process and is responsible for the real-time response of the agent, i.e., for receiving the perception information from the Soccerserver and for sending the adequate action commands to it. It consists of a set of fuzzy controllers. At any given moment, only one fuzzy controller is active and it decides which commands should be sent to the Soccerserver, along with their respective values. This choice is based on the information received from the Soccerserver and it is determined by the active fuzzy controller rules. Each one of the fuzzy controllers available in the agent represents a specific *behavior* and has some associated conditions that specify the situations in which it is effective.

The instinctive level inference engine is implemented in the coordinator process and it is responsible for updating the symbolic variables used by the cognitive level and for chosing the adequate behaviors, i.e., the adequate fuzzy controllers, that should be used in the reactive level in order to achieve a given *goal*. A goal can be achieved by a sequence of reactive behaviors that leads the agent to an intended situation. The choice of this behavior sequence is implemented through a one cycle expert system that chooses, every time the game *state* changes, the most adequate reactive behavior. Each state of the game is defined by a set of conditions that are monitored by the instinctive level. These conditions refer to perception and to the referee messages, and are used in the condition part of the rules, analogously to the reactive level. But on the instinctive level, the conclusion part of the rules are symbolic and are used either to update the symbolic information used in the cognitive level, or to select a reactive level behavior. At each moment, the chosen behavior should perform actions in the direction of the intended goal and should have its associated conditions satisfied by the game state. Once a behavior is chosen, the instinctive level keeps monitoring the conditions associated to this behavior and, if some of them are no more satisfied, it uses its rules to infer a new behavior. If this is impossible, the goal fails and a new goal should be specified. The instinctive level also handles the messages sent by the referee informing a change in the game status.

These changes are treated analogously to the game state changes, they cause the instinctive level to choose a new appropriate behavior.

Finally, the cognitive level inference engine is implemented in the expert process, and it is responsible for determining the local and global goals of the agent. The cognitive level does not have a direct effect over the reactive level, it just chooses the present goal and passes it to the instinctive level. This has the effect of changing the rules of the inference engine in the instinctive level, what indirectly will cause different behaviors to be selected. As long as a goal does not fail or succeed, the cognitive level does not interfere in the game. This idle time is used for strategic planning. This planning consists in the determination of possible future local goals, according to the result of the present one, and in the specification of cooperation requests to achieve global goals. These requests will be handled by the coordinator process and will result in other agents adopting local goals compatible with the intended global goal. The cognitive level is also implemented through an expert system, but this expert system can be much more complex than the instinctive level one, because its response time is much greater.

In the new implementation, the three processes are implemented using the multi-thread programming approach [3]. This technology allows to split a process into parts and to run these parts concurrently. In our case, each process consists of two threads. The first one is responsible for handling the Unix interruption SIGIO, used to inform that a new message has been received by the socket, and by putting this message into the mail box. The other thread, the main thread, is responsible by the process activities. The mutual exclusion between the threads is achieved by using semaphores. This implementation is a concurrent approach to the classical productor/customer problem. It avoids that the main process spend some precious time checking if there is a new message in the socket or not.

The paper is organized as follows. Section 2 describes the reactive level. The instinctive level and cognitive level are presented in Sections 3 and 4. Section 5 presents an example where this new architecture allows the agent to concurrently react to an environment stimulus in real-time and perform more sophisticated tasks like make plan, to establish new goals, open or participate into a cooperation processes, etc. Finally, in Section 6, the conclusions and future works are presented.

2 The Reactive Level

The reactive level inference engine is implemented in the process interface. This process consists of one mailbox, a set of fuzzy controllers, an input filter and an output filter (see Figure 3). The mailbox is responsible for the process message reception. All messages received by the process, including the perception information sent by the Soccerserver, will be stored in the mailbox.

The fuzzy controllers are implemented using a C++ library. This library was designed to aid implementation of fuzzy expert systems or fuzzy controllers im-

plementation, it is called CNCL [13]. Each fuzzy controller is responsible by one reactive agent skill, called *behaviour*. At first, the following set of behaviors was chosen to be implemented into the UFSC-Team agents: *Initialize-Player, Kick-Off-Position, Move-to-Position, Move-to-Ball, Pass-Ball, Kick-to-goal, Dribble-Opponent, Drive-Ball-Fwd, Get-Ball-Control, Tackle, Follow-Opponent, Rounding-Opponent, Watch-Ball* and *Catch-Ball*. The fuzzy controller set associated with each agent depends on which agent group it belongs to: goalie, defensive players, midfielders or attack players. Of course, it does not make a lot of sense for an attack or midfielder player to have a fuzzy controller responsible by catching the ball, or for a goalie to have a fuzzy controller responsible for shooting the ball into the opponent goal.

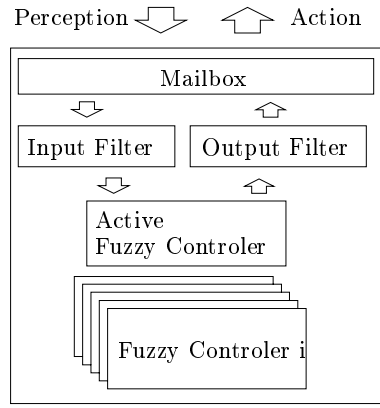

Fig. 3. The interface process

The input filter is responsible for extracting the linguistic variable values, used by the active fuzzy controller, from the perception information sent from the Soccerserver. The output filter is responsible for checking the active fuzzy controller outputs and combining them. The following criteria are observed by the output filter:

- **Null Output:** if dash power output and/or turn moment output present one null output, the respective command is not sent to Soccerserver.
- **Simultaneous turn and dash:** if the fuzzy controller presents simultaneous turn moment output and dash power output, then at first the turn command with turn moment value is sent to Soccerserver. After a 20 ms delay, the dash command with the respective dash power value is sent to the Soccerserver.
- **Kick direction and kick power:** The Kick Direction output and Kick Power output are always joined into the kick command.

Most of the fuzzy controllers have four outputs: kick-direction, kick-power, turn-moment and dash-power. The *Pass-Ball, Kick-to-goal* fuzzy controllers just have the kick-direction and kick-power output and *Move-to-Position* fuzzy controller just have turn-moment and dash-power outputs. The inputs are a set of linguistic variables, depending on which behavior is active. Each fuzzy controller has its own set of linguistic variables and the Input Filter is responsible for extracting from the perception information, the respective values that will be used to set the linguistic variables.

Using fuzzy controllers to implement the reactive level has some advantages. First of all, it is possible to synchronize the agent just adjusting the ratio between input and output, or in other words, adjusting the controller gain. This gain adjustment is made on the fuzzy set wich represent the controller input and the controller output. It is also possible to fine tune, or to get a smooth response adjusting these fuzzy sets. Figure 4 shows the fuzzy set used by the *turn-moment* output and the respective linguistic variable *ball-direction*. Note that in this case the controller gain is $0.56 = \frac{50}{90}$.

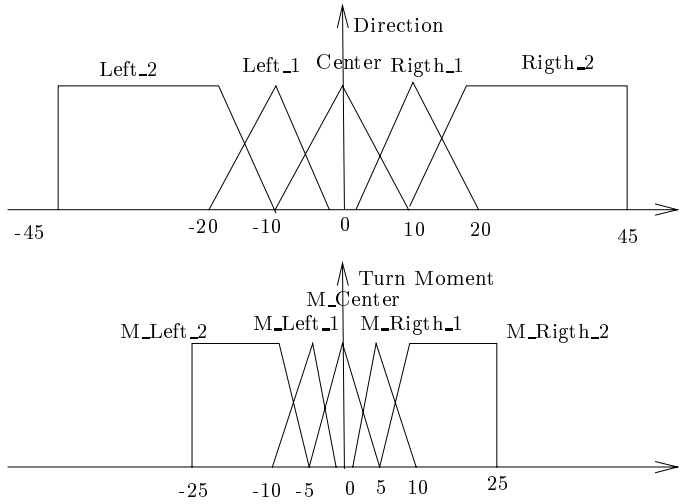

Fig. 4. The Turn Moment Fuzzy Sets

The rules used in the fuzzy controllers can be built in an intuitive way, avoiding the difficult and time consuming task of building a model of the dynamic environment (see Figure 5). It is also possible to use genetic algorithms [5] to improve the fuzzy sets used in the controllers.

Another important advantage of implementing the reactive behavior using fuzzy controllers is that it is possible to ensure that a given fuzzy controller will always be able to satisfy the real-time requirements, because the fuzzy controller is a deterministic system. Beside this, once the active fuzzy controller is the

```
rule_200.add_lhs(new CNFClause(ball_direction, left_2));
rule_200.add_rhs(new CNFClause(turn_moment, m_left_2));

rule_201.add_lhs(new CNFClause(ball_direction, left_1));
rule_201.add_rhs(new CNFClause(turn_moment, m_left_1));

rule_202.add_lhs(new CNFClause(ball_direction, center));
rule_202.add_rhs(new CNFClause(turn_moment, m_center));

rule_203.add_lhs(new CNFClause(ball_direction, right_1));
rule_203.add_rhs(new CNFClause(turn_moment, m_right_1));

rule_204.add_lhs(new CNFClause(ball_direction, right_2));
rule_204.add_rhs(new CNFClause(turn_moment, m_right_2));
```

Fig. 5. The Turn Moment Fuzzy Rules Sets

most appropriated behavior in a given situation, it releases the instinctive and cognitive levels to spend more time into more sophisticated tasks like extracting interesting symbolic features from the perception, making plans, establish goals or participating into cooperation processes.

3 The Instinctive Level

The instinctive level inference engine is implemented in the process coordinator and it is responsible for both the execution of the agent local *goals* and the generation of symbolic information to update the cognitive level knowledge base. It is implemented through a one cycle expert system that chooses, every time the game *state* changes, the most adequate reactive behavior given the current local goal. The current local goal is established by the cognitive level and it determines the set of rules to be used in the inference engine. Each state of the game is defined by a set of conditions on the perception information. These conditions usually depend on some threshold values, that must be determined experimentally.

The inputs to the instinctive level inference engine are the perception information, received from the interface process, and the messages from the referee. The perception information consists of the same synchronous perception information received by the interface from the Soccerserver, but, differently from the reactive level, the instinctive level presents a memory. This memory consists of a buffer, where perception information is stored, and whose initial size is a parameter of the implementation. It makes it possible to choose how many visual information frames can be used in one inference cycle of the inference engine. For example, assuming that the agent has been receiving visual information every 150 ms and that the buffer size is 3, in a given time t, the cycle inference process will take into account the visual information sent at times *t, t-150, t-300.*

The perception information is stored into the *Sync* buffer and the messages received from referee are stored in the *Async* buffer (see Figure 6). Each time

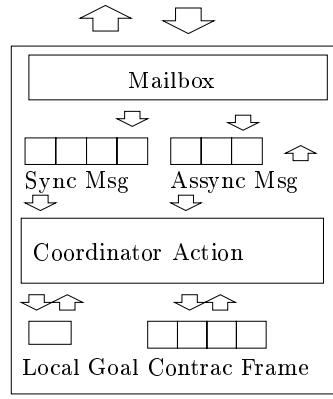

Fig. 6. The coordinator process

one of these buffers is updated or when a new local goal has been received from the cognitive level the expert system is executed. Given the input, the rules are able to recognize changes in the game state. The result of executing the rules can be either the updating of the cognitive level knowledge base or the selection of the most appropriate reactive level fuzzy controller to drive the agent from the current state to the local goal.

Suppose, for example, that the opponent team has the ball control, and our team is performing a defensive play, where the goal is *get-ball-control-back* the current behavior is *Rounding-Opponent*. Also suppose that the opponent player who has the ball control makes a mistake and kicks the ball out of the field. Then the game state is changed to *kick_in_side* and this change will be perceived by a message received from the referee informing about this new game status. In this case, a behavior can be directly selected to be performed by the reactive level, i.e., *Move-to-Ball*, and it also means that the goal *get-ball-control-back* was achieved. The cognitive level will be updated and will generate a new goal. The point here is that in a situation like that, both the new planning and the execution of the new behavior can happen concurrently.

The process coordinator is also responsible for the cooperation. A *Contract Frame Buffer* is provided to store the necessary information involved into a cooperation process that uses the *Dynamic Social Knowledge* Cooperation Strategy [11]. The coordinator process also includes a real-time policy and some management algorithms for distributed system communication, like System Fault Tolerance [6].

4 The Cognitive Level

The cognitive level inference engine is implemented in the expert process. It consists of a symbolic object-oriented knowledge-based system that handles both the

symbolic information received from the instinctive level, and the asynchronous messages received from others UFSC-Team agents. It generates the local goals and the global goals. This knowledge-based system has three knowledge bases: *Dynamic KB, Static KB and Export KB.*

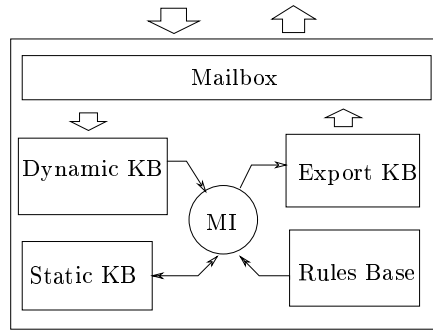

Fig. 7. The expert process

The Dynamic KB is used to store the symbolic information generated by the instinctive level and the asynchronous messages sent by other UFSC-Team agents. The Static KB stores the knowledge that has been inferred by the expert process about the game, the team, the opponent, the agent plans, goals, etc. Both, the Static KB and the Dynamic KB, are used by the inference engine to generate the agent goals. The new facts about the game, the plans and goals are stored into the Static KB. Export KB is used to store the expert process output. Basically this output consists of local goals to be sent to the instinctive level, information to be used in cooperation strategies, or messages to be sent to other UFSC-Team Agents.

Suppose that some symbolic information and/or messages have been received from the coordinator process, followed by a request. This causes the following sequence of actions:

1. The information is stored in the Dynamic KB.
2. The inference engine evaluates both the information stored in the Dynamic KB and the facts stored in the Static KB.
3. The generated new facts, plans and goals are stored into the Static KB.
4. If a new goal is chosen and/or there are some information to be sent to another agent, it is stored into the Export KB.
5. If the Export KB is not empty, its contents are sent to the coordinator.
6. The Dynamic and Export KB are cleaned.
7. A reply to coordinator is sent.

An important feature of this new architecture is that the cognitive level can spend more time planning, establishing goals, etc, once the reactive level and, in

some situations, the instinctive level is responsible for real-time interaction with the environment. The cognitive level also helps the coordinator in the evaluation of the cooperation process to achieve global goals.

5 Example

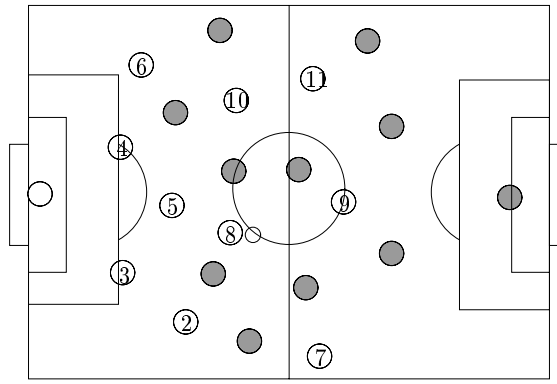

Fig. 8. An example

An example where the proposed concurrent autonomous agent architecture can be useful is presented in this section. Let's assume a situation where the opponent team has the ball control and is performing an offensive play and trying to pass the ball. In this situation the player close to the opponent player, who has the ball control, will have *get-ball-control-back* as its local goal and will be performing the behavior *Rounding-Opponent* into the reactive level. Suppose that the opponent has tried to pass the ball, makes a mistake and the midfielder, player number 8, gets the ball control back. At this situation, shown in figure 8, the agent number 8 instinctive level will recognize that the goal has been achieved and will change the current behavior to *Drive-Ball-Fwd* and inform the cognitive level of the new game state. Then the interface will perform the new behavior and the cognitive level inference engine will choose one of the play-patterns stored into the cognitive level to be performed involving the attack player. This means to select a global goal to be achieved, broadcast this goal to the involved agents and send a local goal, related with the selected global goal to be performed by the instinctive level. Another possibility in this situation is, just after the new behavior begins to be performed by the reactive level, to ask the coordinator to start a cooperation process involving the attack players and midfielders to choose which one of the known global goals is more appropriate for that situation. In the first possibility, the choice is done just taking into account one agent information, i.e., its believes about the environment and about the

other agents. It is not considered whether another player has stamina enough to perform the selected play. In the second possibility, the agent will perform a cooperation process and choose which global goal is more appropriate, taking into account the perception information of all the agents involved in the cooperation process.

6 Conclusions and Future Work

A concurrent autonomous agent architecture to a simulated robot soccer team was presented in this paper. This new architecture explores the concurrent approach to implement an architecture with three levels of decision. It allows the agent to react to an environment stimulus, to make plans, to establish goals and to perform complex agent cooperation strategies concurrently and respecting real-time constraints. It also provides a memory where the perception information can be stored, allowing the agent not just to evaluate the current information, but to evaluate the current and some early past information together. This allows information about the object movement to be handled and the current and past information to be used to make inferences about the environment.

Some real-time policy is also provided to ensure that the agent is handling the newest information, like dedicated buffer to perception information. If it is assumed that the size of the buffer is two, it is sure that the two last perceptual information are stored in that buffer. Also it is possible to choose what kind of information will be handled first, perceptual or asynchronous message. Beside this, some implementation effort was done to handle the received messages, like multi-threads programming approach associated with the Unix SIGIO interrupt, and represented a significant improvement in the real-time response.

One further advantage of the proposed architecture is that the numerical parameters of the implementation are partitioned into two subsets: the limits of the fuzzy sets used in the reactive level controllers and the thresholds used by the instinctive level to calculate the logical values of the symbolic variables used in the cognitive level. In the future, we intend to use optimization methods, such as Reinforcement Learning [8] and Genetic Algorithms [5], to independently improve these two set of parameters. These parameters, because they refer only to local behaviors, can be improved using simplified situations, whith few players.

This architecture has been used to implement the UFSC-Team and it is already integrated to the partial implementation of the environment to build cognitive multi-agent system under real-time restriction, called Expert-Coop++. It will allow the UFSC-Team to employ complex cooperation strategies that use both perception information and some communication among the agents involved into the cooperation process. In a near future some cooperation strategies will be implemented and evaluated in the UFSC-Team.

Acknowledgments

The authors express their thanks to the hospitality of the staff of the IAKS (Universität Karlsruhe) where part of this work was developed. The authors are also grateful to the "Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (Capes)", a Brazilian research support agency, for the partial support of this project.

References

1. Andrews, G. R.: Concurrent Programming: Principles and Practice. The Benjamim/Cumming Publishing Co. (1991)
2. Allen, B.P.: Case-Based Reasoning: Business Applications Communications of ACM (1994), March, vol. 37, Nr 3, pages 40-42
3. SunSoft: Multi Threads Programming Guide. Sun Microsystems Inc. (1994).
4. Bittencourt, G.: In the Quest of the Missing Link. Proceedings of IJCAI 15, Nagoya, Japan, August 23-29 Morgan Kaufmann (ISBN 1-55860-480-4), (1997), pages 310-315.
5. Davies, L.: Handbook of Genetic Algorithms. CVan Nostrand Reinhold, New York (1991)
6. Jalote, P.: Fault Tolerance in Distributed System. PTR Prentice Hall, Englewood Cliffs, New Jersey (1994).
7. Kitano,H.: RoboCup: The Robot World Cup Initiative. in Proc. of The First International Conference on Autonomous Agent (Agents'97). Marina del Ray, The ACM Press, (1997).
8. Kaelbling, L. P. and Littman, M. L. and Moore, A.W.: Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research, (1996), vol 4, pages 237-285 .
9. Bittencourt, G. and Costa, A. C. P. L. da: Expert-Coop: An Environment for Cognitive Multi-Agent Systems in pre-printers IFAC/IFIP MCPL'97, Conference on Management and Control of Production and Logistics, vol 2, (1997), pages 492-497.
10. Costa, A. C. P. L. da and Bittencourt, G.: Parla: A Cooperation Language for Cognitive Multi-Agent Systems. EPIA'97, 8th Portuguese Conference of Artificial Inteligence, Spring-Verlag, Lecture Notes in Artificial Inteligence vol 1323, (1997), pages 207-215. Production and Logistics, vol 2, (1997), pages 492-497.
11. Costa, A. C. P. L. da and Bittencourt, G.: Dynamic Social Knowledge: A Cooperation Strategie for Cognitive Multi-Agent Systems. Third International Conference on Multi-Agent Systems, ICMAS'98,Paris, France, July 2-7 (1998), IEEE Computer Society, pages 415-416.
12. Costa, A. C. P. L. da and Bittencourt, G.: UFSC-Team: A Cognitive Multi-Agent Approach to the RoboCup'98 Simulator League. RoboCup´98 Workshop - Team description (1998), pages 371-376.
13. Junius, M. and Stepple, M.: CNCL Reference Manual. Universität Aachen, (1997), http://www.comnets.rwth-aachen.de/cnroot_engl.html