Actas do Encontro Científico
3º Festival Nacional de Robótica - ROBOTICA2003
Lisboa, 9 de Maio de 2003.

LOGIC BASED DISTRIBUTED DECISION SYSTEM FOR A MULTI-ROBOT TEAM [1]

Miguel Arroz, Vasco Pires, Luis Custódio

Instituto de Sistemas e Robótica
Instituto Superior Técnico
Av. Rovisco Pais, 1049-001 Lisboa, Portugal
{mbsa,vmicp}@rnl.ist.utl.pt, lmmc@isr.ist.utl.pt

Abstract: In this paper a decision system for a team, constituted by several robots is described. The system is distributed and uses logic to choose the most suitable action in order to accomplish a pre-determined goal. It also supports cooperation and machine learning.

Keywords: Logic-Based Decision System, Prolog, Golog, Hybrid Architecture, RoboCup Middle-Size League.

1. INTRODUCTION

Nowadays, we can see an increasing amount of robotic systems for an also increasing number of purposes. From robots assembling automobiles in a production line, to robots guiding visitors through out a museum, the possibilities are endless. Some of these applications require, not just a single robot, but a group or team of robots that must work together to accomplish the goal.

Robotics has improved in many ways. Mechanics are becoming faster, more precise, reliable and durable. Electronics provides greater autonomy, reliable sensors, and faster processing. Artificial vision endows robots with the ability to detect objects in the world by just *looking* at them with video cameras, with no special extra sensors. But, even with all this great new technology, a big problem remains: how to *decide* what the robot should do? A robotic arm can pick up a glass of water without spilling it, but *when* should it do that? *Why* should it do that? Will picking up the glass help to accomplish the goal?

This is why the link between Robotics and Artificial Intelligence (AI) is becoming more relevant every day. Robotics provides the means to interact with the real world. AI allows a robot to decide over that world. Most robots rely mainly on reactive decision systems. Those systems are based on simple reactions to external or internal events, and implemented by inflexible tools, like state machines or decision trees. Robots using this kind of tools usually show very primitive behaviors, and are not able to accomplish non-trivial goals on complex, dynamic, and incomplete domains. On the other side, we have deliberative systems, which are able to make decisions based on more sophisticated tools, like a logic engine. These systems have many advantages over reactive ones. They have more expressive power, in the sense that they allow us to model the world using a set of facts and rules, instead of modelling rigid behaviours, like in a state machine. Those rules are then used by the system to make a decision based on the current world status and previously stored information. The system may even act and decide in a way that was not predicted by the programmers, but that is actually valid and efficient in order to achieve the goal.

[1] Work supported by the FCT “Programa Operacional Sociedade de Informação (POSI)” in the frame of QCA III.

Another advantage is that the deliberative system is not event driven, as a reactive system is. This means that a reactive system changes its state only when some pre-determined event happens. A deliberative system may be always analysing the world, and making decisions. It does not need an event to happen in order to change its behaviour. Or, it may behave in different ways when facing the same world status. So, a deliberative system allows more flexibility in the behaviour modelling.

Of course, this advantage does not come for free. Deliberative systems are computationally heavy. A logic engine may need hours to make a single decision. The CPU and memory usage cannot be underestimated for this kind of decision systems, even more in real-time domains. A robot cannot be stopped for hours (or even minutes, and, sometimes, not even seconds) to *think* what to do next. It must be permanently making decisions, and acting according to them.

How can we take advantage of both kinds of systems? How can we make intelligent decisions, and, at the same time, guarantee that the robot will act in real time? A possible solution is an hybrid architecture (Coelho and Paiva n.d.). A typical hybrid architecture is illustrated in Fig. 1. With this architecture, we have both systems working in parallel, independently. The system uses, normally, the decisions made by the deliberative component. But, when the deliberative component takes too long to decide, it uses the decision made by the reactive component. That decision is not as good as the one provided by the deliberative component, but it is better than doing nothing.

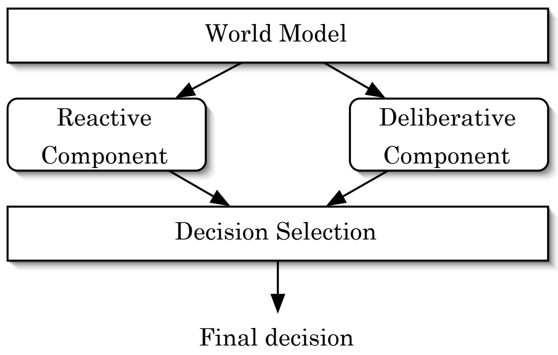


Fig. 1. Hybrid Architecture

The paper structure is as follows: in section 2 the RoboCup domain is described; section 3 presents the proposed architecture and a detailed description of its components, and in section 4 we present some results and conclusions.

2. ROBOCUP DOMAIN

Our domain is the middle-size RoboCup soccer league [2]. A middle-size league team is constituted by four robots (generally, three field players and one goal-keeper). The robots diameter is about 40 cm, and the game is played on an (approximately) 10x5 meters field. The robots may communicate via Wireless Ethernet, and may also send information to a computer (external to the game) for monitoring and debugging purposes. According to the RoboCup rules, the external computer may send data or commands to robots, but the fundamental law of our team is that robots should be completely autonomous. So, our robotic team only relies on information gathered from onboard sensors.

The domain is very incomplete: it is really hard for a robot to have a complete representation of the world, because sensors are not perfect. For example, images acquired by the cameras usually have noise, so the data robots get from them have errors. The domain is also very dynamic: all robots are moving, the ball keeps moving around the field, so there are a lot of things happening at the same time.

All these characteristics make the middle-size RoboCup league a very interesting and challenging domain for AI. In the next section we shall describe a software architecture developed for handling some of these problems. It is being implemented on the ISocRob [3] team, the team from the Instituto de Sistemas e Robótica (ISR), at Instituto Superior Técnico, Lisbon.

3. ARCHITECTURE

The hybrid architecture developed for our team, that enables us to use deliberation and reactiveness, is presented in Fig. 2.

This architecture is composed by several components, from which the most important ones are: World Representation, Reactive Component, Deliberative Component and Behavior Selection.

3.1 World Representation

The World Representation Component (WRC) is responsible to build a world model using sensorial data. From the sensory inputs and the static information about the game, the WRC builds the game model, that consists of basic information, like ball position and players postures, and advanced information such as cooperation decisions. The

[2] http://www.robocup.org
[3] http://socrob.isr.ist.utl.pt

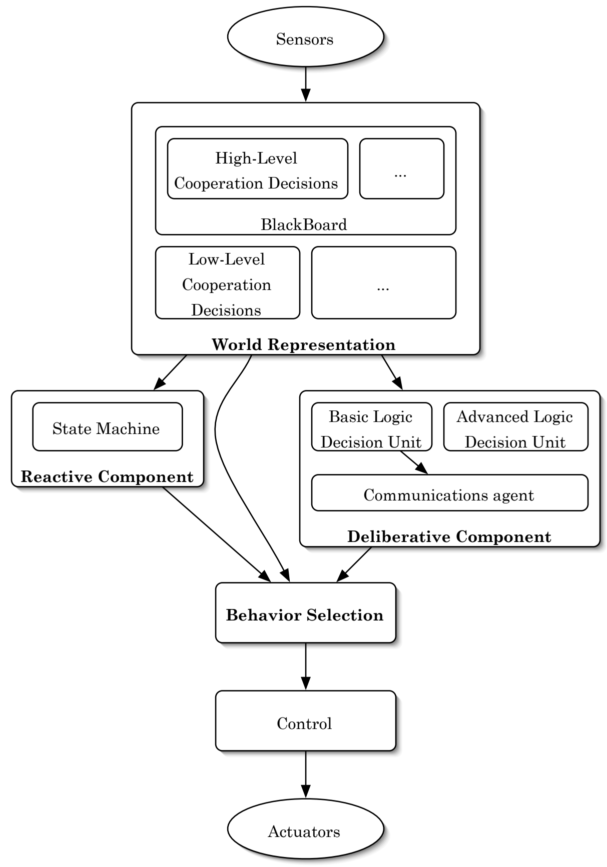


Fig. 2. Hybrid Architecture.

variables used to define the world model are stored in a Blackboard. The Blackboard is a data pool accessible by several components, used to share data and exchange messages among them. Traditional blackboards are implemented by shared memories and daemons that awake in response to events such as the update of some particular data slot, so as to inform the components requiring that data updated. In our implementation, the Blackboard consists, within each individual robot, of a shared memory among the different components, organised in data slots corresponding to relevant information (e.g. ball position, goal position), accessible through *data-keys*. Some variables of the blackboard are *local*, meaning that the associated information is only relevant for that robot, but others are *global*, so their updates must be broadcasted to the other teammates (e.g., the ball position) (Lima 2002).

The cooperation is divided in a high-level cooperation and a low-level cooperation. The former one is stored in the Blackboard, and consists of *Group-Level* and *Team-Level Tactics*, that can be viewed as analogues of the coach's directives in real soccer. The *Group Level Tactics* defines tactical parameters for the different player groups: defense, mid-field and attack. For instance, a good defensive tactic is to form a defensive line with the goalkeeper to block all paths to our goal. The *Team-Level Tactics* set general tactical conditions of the whole team. Parameters as basic formation, e.g. 2 defenders - 1 attacker, if we are in a defensive play, or 1 defender - 2 attackers, if we are in an offensive play. The low-level cooperation is outside the blackboard because it is a commitment between the robots that are involved in a cooperative action, e.g. when a robot tells another teammate to move to a certain position, in order to be able to receive a pass. It's necessary to have a communication method between all the robots, so they can exchange messages among them. We pretend to use an Agent Communication Language (ACL)(FIPA 2002), allowing us to use a standard and highly flexible message format.

3.2 Reactive Component

The purpose of having a reactive component is to quickly settle the next basic behavior to execute. If the deliberative component takes too long to decide a behavior, the reactive component can take its place and define the next action to execute. This idea is inspired on the way humans make decisions when there is no time to think rationally, e.g. when a glass of water is falling, we don't think, we react and try to catch it (Sadio *et al.* 2001) .

Many different reactive systems have been developed, e.g. state machines (Lima 2002), decisions trees (see the reactive system from (Dylla *et al.* n.d.)), and neural networks , but all of them are hard to modify or limited. These limitations are due to the weak expressiveness inherent to reactive components, e.g. a state machine is a static tool, where it is hard to modify some behavior during the game. For example, if the goalkeeper is at the middle of the goal, and catches the ball when it was kicked to the right side of the goal, then next time the robot probably should kick it to the left. With state machines we cannot easily implement this kind of dynamic behavior switching. Our aim is not to develop a new reactive component, but to build an architecture that integrates the speed of a reactive system with the expressiveness of a deliberative system. For the ISocRob team the reactive component is for the time being a finite state machine, illustrated in Fig. 3, because it's already implemented and working well.

3.3 Deliberative Component

The deliberative part is the main component in this work, where all logical decisions are made. This component is divided in two decisions units, a basic decision unit and an advanced decision unit. The basic decision unit tries to quickly settle

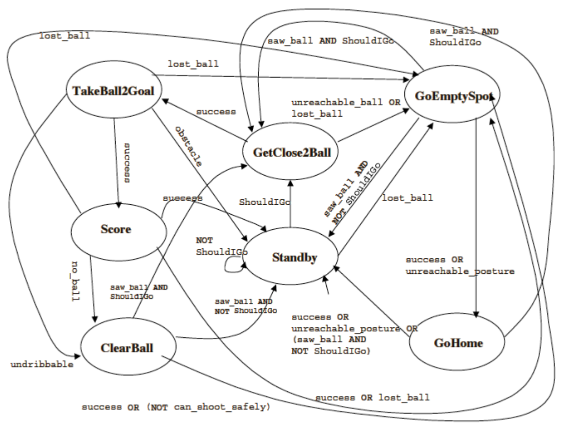


Fig. 3. The current state machine.

the next behavior to execute, using a basic logical reasoning. The difference between this unit and the reactive one is that decisions took by this component are based on a simple and expectedly fast deliberative system, based on situational calculus. Furthermore we intend to add basic machine learning to this unit that will make it less reactive. One reason for its existence comes from the fact that a logical approximation gives us expressiveness (we can make a more abstract decision systems, where we do not have a predefined sequence of states or behaviors), flexibility (this comes from abstraction) and easier evolution. Besides that we expect it to take the role of the reactive component in a near future, if it proves to be fast enough to make its decisions in real time. The advance decision unit is computationally more heavy. It generates sequences of behaviors (plans) that can be executed by the robot. Besides that it can be used to plan certain non-tactical strategies, e.g. attack from the left side, if the left defender is not working properly. Due to its complexity this unit can take some time to compute a result. An initial implementation of the basic decision unit was done using the Prolog programming language whereas the advanced decision unit will be implemented using an action programming language called Golog (Levesque *et al.* 1997). Golog is based on the Situation Calculus (McCarthy 1963), allows to easily create high level behaviors, and express them as logical sentences, with pre-conditions and post-conditions.

The basic logic component unit uses simple rules written in prolog in order to select the right behavior. Each rule has a set of predicates, that ought to be true for the corresponding behavior to be executed. Rules may have priority one over another, this priority defined by the order they appear on the input file. Rules that come first have priority over the ones that come later. In Fig. 4 we can see an example of those rules. In this rule we can see the robot assumes the score behavior when the game is running, the robot can see the ball, it has the ball, and finally the robot is near the opposite goal. The complete set of rules currently implemented in the basic logic decision unit is presented in Fig. 10 (at the end of the document).

```
basicBehaviour( score, 0 ) :-
        gameRunning(1),
        \+ state_finished( score ),
        vision_seeball,
        has_ball,
        near_goal.
```

Fig. 4. Score rule.

The predicates are also rules programmed in prolog (although some of them call C functions) and are very easy to write. As an example, we show in the Fig. 5 the predicate used to decide if the robot has or hasn't the ball with him. This predicate is true when the ball is at most at a predefined distance (0.38 meters in this case) and between an inside a predefined angle. These values are currently static, but in the future they may become dynamic, based on machine learning.

```
has_ball :-
        vision_ball_dist( Dist ),
        Dist < 0.38,
        vision_ball_angle( Angle ),
        Angle > -20,
        Angle < 20.
```

Fig. 5. has_ball predicate.

The behaviors selected by these components are executed by the control component.

3.4 Behavior Selection

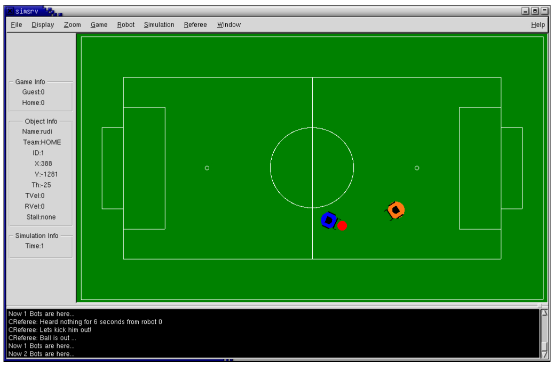
Behaviour Selection (BS) is the component that makes the final decision. It chooses among the decisions produced by the reactive component, the basic logic decision unit, the advanced logic decision unit and requests from other robots, what behaviour will be executed. The choice is based on heuristic functions attached to each of the possible decisions produced by each unit. The BS component simply executes the heuristic functions on the decisions themselves.

Why do the heuristic functions come attached to decisions? Why don't we just calculate an heuristic value when the decision is made, and act according to the decision with the highest heuristic value? Although some decisions are to be executed immediately (like the decisions the RC takes), there are others (like the plans the DC generates) that are executed during some time. As the world evolves, the plan may remain valid and useful, or become outdated or pointless. This is why the heuristic function is important: it must be executed at regular time intervals, to make sure the heuristic value of that decision is still the higher one.

At the end, the BS selects the most suitable behavior using the heuristic functions, and commands the control unit to assume the behavior. This component uses several very small state machines (about two or three states each) to implement the behaviors. It might happen that in the future the state machines are replaced by a more flexible implementation, that allows high level components to have a more direct and precise control over the robot.

4. RESULTS AND CONCLUSIONS

In order to work in a more efficient way, we have been adapting the robots software to a real robot simulator(Kleiner and Buchheim 2003). This simulator was specially developed to reproduce the main characteristics of some real robot models (as the Super Scout - Nomadic robots which constitute the ISocRob Team), and their usage in a RoboCup middle-size game environment. It works in a server/client fashion (the server is the simulator itself, the clients are the simulated robots). Also, the simulator has a graphical user interface that allows users to follow the game, and manually move objects on the field. A screenshot of the simulator is presented on the Fig. 6.



Guest:0
Home:0

Name:rudi
Team:HOME
ID:1
X:388
Y:-1281
Th:-25
TVel:0
RVel:0
Stall:none

Time:1

```
Now 1 Bots are here...
CReferee: Heard nothing for 6 seconds from robot 0
CReferee: Lets kick him out!
CReferee: Ball is out ...
Now 1 Bots are here...
Now 2 Bots are here...
```

Fig. 6. Simulator screenshot.

Currently, we have developed a prototype of the basic logic decision unit. In order to compare this basic logic decision unit with the reactive system currently used (the statement shown in Fig. 3), we have tested both using the simulator. To do that we used the following test: the robots starts facing the opposite goal, as illustrated on Fig. 7.

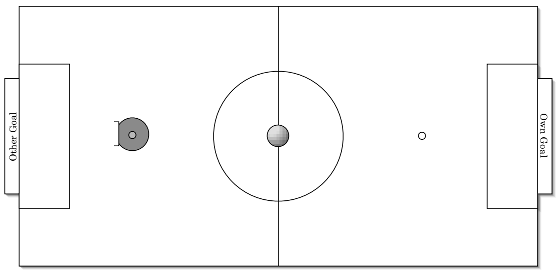


Fig. 7. Initial object positions.

The robot must go back to the middle of the field, get the ball, return to the opposite goal and score.

We applied this test ten times to each system (logic based decision system and the original state-machine decision system). The results are on the tables 1 and 2. We illustrate also the best sample of both systems, in Fig. 8 and 9. The graphics show the path followed by the robot, and the positions occupied by the ball during the test, at regular time intervals.

Table 1. Results for the logic based decision system.

Sample	Ball Losses	Path Length	Time	Ball Out
1	2	10.95	50	0
2	1	9.08	26	0
3	0	7.62	22	0
4	0	9.15	22	0
5	2	11.41	38	1
6	1	9.64	37	1
7	1	10.45	37	0
8	1	8.95	32	0
9	0	7.41	18	0
10	0	8.12	23	0
Average	0.8	9.28	30.5	0.2

Table 2. Results for the state machine based decision system.

Sample	Ball Losses	Path Length	Time	Ball Out
1	1	12.17	34	0
2	0	7.16	23	0
3	1	10.00	35	0
4	1	9.70	32	0
5	1	10.71	43	0
6	1	9.91	33	0
7	0	6.87	20	0
8	1	10.14	41	0
9	0	7.00	19	0
10	1	10.29	50	1
Average	0.7	9.40	33	0.1

Not surprisingly the results obtained are similar for both systems, since the low-level control is the same. For the robot performance on this test, the low-level control is much more important than which tool is used for behavior switching. However, from the results we observe a tendency: the logic based robot was able to score in less time than the state-machine one. This is related to the fact that the deliberative system is not event

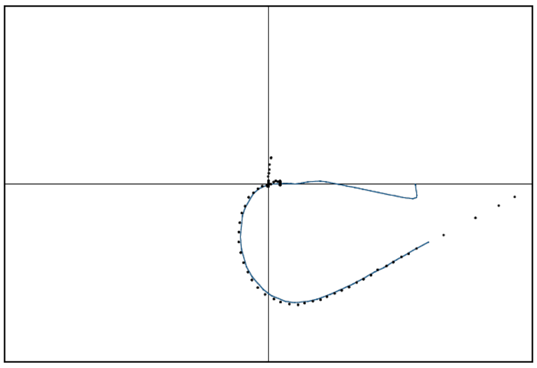


Fig. 8. Logic based robot path.

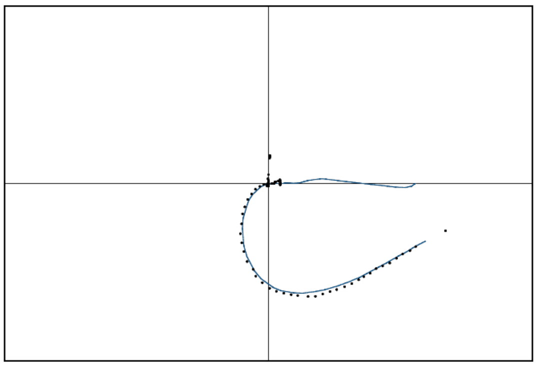


Fig. 9. State-machine based robot path.

driven. On the reactive system, the state machine must wait for an event (usually, a signal from the control level announcing that the previously selected behaviour has finished) in order to change its state. This does not happen on the deliberative system, as it permanently analyses the world and chooses the best thing to do. It does not have to wait for anything to decide the execution of a new behaviour. Due to this, the logic based robot may switch behaviors with a better timing, and be globally faster than the reactive robot. However, more tests involving more complex game situations should be performed in order to confirm this tendency. For instance, situations with more than one robot, with an adversary team or involving some simple cooperative actions would be useful to compare than two behaviors switching tools.

Also, it seems that processing time is not a problem, at least for now. The logic unit works really fast, and the decisions are made with no delay or relevant processing time. We believe that we are able to write more complex rules, while keeping enough speed for real-time decisions. Although we intend to keep the state machine as the reactive component for the reasons stated above, the basic logic decision unit will probably be fast enough to replace it completely.

This work has also revealed how fast and simple is to model the robot high-level behaviour using a logic based system. With a few lines of code (no more than a printed page), we could make the robot behave as if it was being controlled by the state machine, even a little better, as the results show. This logic-based approach will allow us to design more complex behaviors and use more information (e.g., field zone where the robotics, other robots postures, etc) with much less effort than if the same is to be done with state-machines. Moreover the code remains easy to read and modify, contrariwise to what happens with the state machine.

```
basicBehaviour( goHome, Home ) :-
        gameRunning( 1 ),
        inside_goal,
        home( Home ).

basicBehaviour( goHome, Home ) :-
        gameRunning( 1 ),
        get_state( St ),
        St = goHome,
        \+ state_finished( goHome ),
        home( Home ).

basicBehaviour( goEmptySpot, EmptySpot ) :-
        gameRunning(1),
        \+ state_finished( goEmptySpot ),
        \+ vision_seeball,
        emptySpot( EmptySpot ).

basicBehaviour( score, 0 ) :-
        gameRunning(1),
        \+ state_finished( score ),
        vision_seeball,
        has_ball,
        near_goal.

basicBehaviour( takeBall2Goal, 0 ) :-
        gameRunning( 1 ),
        \+ state_finished( takeBall2Goal ),
        vision_seeball,
        has_ball.

basicBehaviour( getClose2Ball, 0 ) :-
        gameRunning( 1 ),
        \+ state_failure( getClose2Ball ),
        vision_seeball,
        \+ has_ball.

basicBehaviour( standBy, 0 ) :-
        gameRunning(0).

basicBehaviour( standBy, 0 ).
```

Fig. 10. Current Basic Logic Unit code.

ACKNOWLEDGMENTS

This work was produced with the support of Prof. Pedro Lima whose help and guidance was a valuable contribution.

REFERENCES

Coelho, H. and A. Paiva (n.d.). A mente e o mundo lá fora.

Dylla, F., A. Ferrein and G. Lakemeyer (n.d.). Acting and deliberating using golog in robotic soccer - a hybrid architecture.

FIPA (2002). Fipa acl message structure specification. Technical report. Foundation for Intelligent Physical Agents.

Kleiner, A. and T. Buchheim (2003). Simsrv, a robocup simulator.

Levesque, H., R. Reiter, Y. Lesprance, F. Lin and R. Scherl (1997). Golog: A logic programming language for dynamic domains. *Journal of Logic Programming.*

Lima, P. (2002). Current status of the socrob project. Technical report. Instituto de Sistemas e Robótica.

McCarthy, J. (1963). Situations, actions and causal laws. Technical report. Standford University.

Sadio, R., G. Tavares, R. Ventura and L. Custódio (2001). An emotion-based agent architecture application with real robots. *AAAI Fall Symposium.*