

KULRoT 1999 Team Description

KULRoT

Kurt Driessens, Nico Jacobs, Bert Robben

KD, NJ, BR: K.U.Leuven

Abstract. *This paper gives an overview of the KULRoT 99 team. This team is based on our last year's team. The major problem we experienced with KULRoT 98 was a slow reaction time. The cause of this problem was related to bad synchronization and an insufficient notion of time. To tackle this problem, we switched to a new architecture that gives each soccer player a better notion of time. Preliminary results already show a large improvement in reaction time. In this position paper, we describe this architecture and briefly mention some plans for future work.*

1 Introduction

The 1999 KULRoT RoboCup soccer team is the successor of the KULRoT 98 team. This latter team was based on a reactive agent architecture. This team competed in the '98 RoboCup world cup but was also used in experiments to learn strategies using a genetic algorithm [4], experiments on verification and validation of multi-agent systems [3] and data mining research [1].

The major problem we experienced with KULRoT 98 was however a slow reaction time. The cause of this problem was related to insufficient synchronization between the changing environment and the soccer players. To solve this problem, we created an improved architecture that gives a player a much better notion of time. This allows the player to adjust his actions and plans much better to occurring events. First in section 2 we discuss last year's approach and the problems caused by this. In section 3 we describe the agenda based approach. We also mention some possible extension in section 4 before we conclude.

2 Synchronization problems with last year's team

The KULRoT team is implemented in Java. A player was implemented as two cooperating components: a sensor and a brain. The sensor component processed the incoming data: it listened to the dedicated port, parsed the incoming information, translated the relative information into absolute coordinates and updated the data structures. The other component (brain

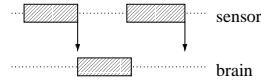


Figure 1: Synchronization problem

component) decided which high level action to perform, translated this into low level actions (turn, kick, dash or say) and sent this to the Soccer Server (see [3, 2] for more information).

A difficult problem that needs to be solved is the synchronization of these components. The sensor is triggered by incoming information sent by the soccer server. The brain is in principle independent of the sensor and can plan whenever it wants. When these two components are not properly coordinated, undesired behaviour occurs.

For instance, when the brain is totally independent of the sensor, it might see information that is completely out-of-date. Imagine a player looking for the ball. The player regularly checks whether he sees the ball and if not turns 90 degrees. If there is no synchronisation between checking the visual information for the ball and sending the turn command, the player may be turning forever. So the brain needs to know when the information about the world is updated and it is time to reflect on his further actions

A simple solution to this is to let the sensor notify the brain after it has processed all input. However in this situation the problem arises that the brain doesn't know whether the information it receives already reflects the changes resulting from the previous action. Consider figure 1 which shows when components are active (rectangle) or inactive. After the sensor first notifies the brain (indicated by the arrow), the brain decides upon which action to take and sends the appropriate commands to the server. Later on, the brain gets another notification from the sensor, but it doesn't know whether the new information reflects the situation before, during or after the execution of the previous action send to the soccer server.

One can extend this schema by letting the sensor inform the brain when it receives new data from the soccer server as well as when it has processed the data. The brain then knows that when the sensor started to process information it received before the brain had sent its last action, the new information still doesn't reflect the changes resulting from the last action. However, when the sensor receives information from the soccer server after the brain has sent its last action, one still doesn't know whether the new information reflects the state of the world after the execution of the last action because one doesn't know whether the server has already completed the last action.

The lasting problem is that with the above schema one never knows whether the information one receives already incorporates the last action sent to the server. To solve this problem we let the brain estimate the action execution time. After the last action was sent to the server, the brain estimates how long it will take the server to execute this action (say till timestamp T), and waits until the sensor has processed new world information that originated after timestamp T before deciding upon which action to perform next.

This solution was used in the KULRoT 1998 team. This approach is a very conservative approach. Our agents always waited till they were certain they had information about the world that incorporated their last action. As a result of this, our agents always decided upon taking 'reasonable actions'

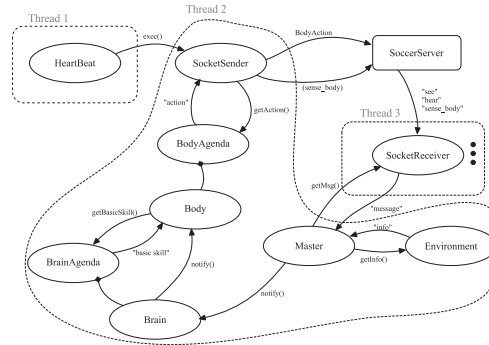


Figure 2: Structure of the 1999 KULRoT agent

(no ‘strange behavior’ due to wrong world information) but were extremely slow due to ‘oversynchronisation’. Our approach didn’t allow us to use the benefit of the the possibility to act more often than world information is received. Moreover, in some situations the agent didn’t act for multiple simulation cycles.

As a result of this, we decided upon using a different approach for the 1999 team. This agenda based approach is explained in the next paragraph.

3 Agenda Based Approach

The 1999 KULRoT SoccerAgent was build with a new and improved design. Instead of the decide-execute loop of last years agent, where the SoccerAgent would calculate the correct action and then wait until it was executed to decide on the next action, the SoccerAgent now has the ability to generate plans. These plans can be based on the agent’s current knowledge about the world state as well as on the plan already made but still to be executed. To be able to revise or remove these plans as necessary to incorporate unpredicted changes in the agent’s environment, these plans are stored in an agenda and executed concurrently with the generation of plans.

The way this is implemented is illustrated by figure 2. The two main new modules in this design are the Brain and Body agendas. Introducing two agendas in our SoccerAgent Design allows us to use the layered approach to soccer skills also employed by [5]

The Brain-agenda holds the high level planning of the agent’s actions. This agenda is re-evaluated and, if necessary, altered every time new visual information is available. This new information allows the agent to re-evaluate the current world state and adjust its plan to incorporate this new knowledge. The actions put on the Brain-agenda include high level soccer skills such as dribble, shoot, and pass to a team-mate. The Body-module translates the high level actions that are on the Brain-agenda into plans consisting of the commands available in the soccerserver. It build these plans taking into account the current world state and actions which are already planned but not yet executed.

In the current implementation, the Body-module is awakened after the inspection and possible adjustment of the Brain-agenda, so that it is able to adjust the Body-agenda to incorporate the changes made. One could synchronize the Body-module with the incoming Body-Sensing messages as well, to be able to adjust the Body agenda to the information available in

these messages. This is not yet implemented. The instruction-execution-module is implemented completely separate from the planning module described above. Every instruction cycle, the execution module takes the top command from the Body-agenda and sends it to the soccerserver. When the agenda is empty, i.e. when the agent thinks no action is necessary or appropriate at that time, it does nothing. The last action executed is not removed from the agenda until its effect has been observed in the world state. This allows the agent to incorporate actions which are already sent to the soccerserver, but of which it hasn't observed any consequences yet.

4 Extending the system

The use of this agenda-design can be augmented to a higher action-abstraction level, or to put in in RoboCup terms to a higher level of strategy specification. The agenda provides a simple way to incorporate special plays into a soccer team. By representing the role of each player in a special play as a plan of actions to be executed, an agent can quickly alter its behavior to suit special world situations. The real translation of these actions into executable commands will then be handled by the low level planner which is able to cope with small environment changes or the randomness added by the soccerserver without having to alter the high level planning. This insures that the special plays will not fail or be abandoned whenever a small unexpected change occurs in the environment.

The agenda model also allows the SoccerAgent to divide its time between high level reasoning and low level reasoning in a more controlled manner than in the old design. This allows the new agent to adjust the depth of its reasoning to the processing time available. More processing time allows for more elaborate planning, while little processing time will have no effect on the rate of execution of the low level commands.

5 Conclusion and future work

In this paper we described the 1999 KULRoT team. The main difference with last year's team is the switch from a reactive based agent to a planning agent which uses two agenda's: one for low level actions, one for high level actions. In the future we plan to test this architecture with appropriate strategies that use the planning capabilities that this architecture offers.

Acknowledgements: The authors wish to thank Serge Demarre and Kris Steegmans for refining and implementing the current agenda based planning system. We also like to thank Luc De Raedt for his stimulating ideas. Nico Jacobs is financed by a specialisation grant of the Flemish Institute for the promotion of scientific and technological research in the industry (IWT).

References

- [1] H. Blockeel, L. De Raedt, N. Jacobs, and B. Demoen. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, 3(1), 1999.
- [2] N. Cossement. Robocup: developing low level skills. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1998.

- [3] K. Driessens, N. Jacobs, N. Cossement, P. Monsieurs, and L. De Raedt. Inductive verification and validation of the KULRoT RoboCup team. In *Proceedings of the second RoboCup Workshop*, pages 135–150, 1998.
- [4] P. Monsieurs. Developing high level skills for robocup. Master’s thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1998.
- [5] P. Stone and M. Veloso. A layered approach to learning client behaviours in the robocup soccer server. *Applied Artificial Intelligence*, 1998.