RoboLog Koblenz 2001*

Jan Murray, Oliver Obst, and Frieder Stolzenburg

Universität Koblenz-Landau, AI research group
Rheinau 1, D–56075 Koblenz, GERMANY
{murray,fruit,stolzen}@uni-koblenz.de

1 Introduction

Outline. A formalism for the specification of multiagent systems should be expressive enough to model not only the behavior of one single agent, but also the collaboration among several agents and the influences caused by external events. For this, *state machines* [4] seem to provide an adequate means. Therefore, the approach of the team *RoboLog Koblenz 2001* employs techniques from software engineering and artificial intelligence research by using UML statecharts and implementing them systematically with *logic and deduction* in Prolog [3].

The current work concentrates on formal agent design. The decision process of soccer agents can be made more flexible by introducing *utility functions* for rational behavior as proposed in [5]. Furthermore, it is desirable to be able to check whether the multiagent system satisfies certain interesting properties. Therefore, the formalism should also allow the verification or the formal analysis of multiagent systems, e.g. by *model checking*, as described in [7].

The RoboLog team. The RoboLog team participated in the simulator competitions in 1999 (Stockholm) and 2000 (Melbourne). 3 people, Jan Murray, Oliver Obst and Frieder Stolzenburg (team leader), form the core of the team. There are currently 6 additional members, namely Joschka Bödecker, Björn Bremer, Marco Dettori, Marion Levelink, Jana Lind, and Karsten Sturm. However, most of them joined the group very recently. As in previous years [8], the team is implemented in two parts. The kernel, hosting the soccer server interface and low-level functions, is implemented in *C++*, while the control program for the team behavior is written in *Prolog*.

In Section 2, we describe our approach with an explicit state machine and its concrete implementation [2]. This is part of the current team, later on this shall lead to verification [7] and more sophisticated decision making [5] of agent systems. Building agents for a scenario such as the RoboCup also requires the careful and efficient programming of low-level facilities (see Section 3). Ball interception and marking are important features (already efficiently incorporated into our team), but also the ability of more abstract, qualitative reasoning.

* This research is partially supported by the grants *Fu 263/6-1* and *Fu 263/8-1* from the German research foundation *DFG*.

A. Birk, S. Coradeschi, and S. Tadokoro (Eds.): RoboCup 2001, LNAI 2377, pp. 526–530, 2002.

2 Declarative Agent Design

Structured State Machines. Statecharts are a part of UML [4] and a well accepted means to specify dynamic behavior of software systems. They can be described in a rigorously formal manner, allowing for flexible specification, implementation and analysis of multiagent systems [7]. In statecharts, states are connected via transitions with conditions and actions annotated. Since states may be simple, composite or concurrent, the behavior of agents or their state machines, respectively, cannot be described by sequences of simple states (as for finite automata), but of configurations.

A *configuration* c is a rooted tree of states, where the root node is the topmost initial state of the overall state machine. A configuration must be completed by the following procedure: if there is a leaf node in c labeled with a composite state s, then the initial state of s is introduced as immediate successor of s; if there is a leaf node in c labeled with a concurrent state s, then the tree branches at this point. In the current implementation of our team, an explicit state machine is built in. It processes the transitions, performing micro-steps in this case. Several transitions can be executed in parallel if they stem from concurrent regions, forming a macro-step [6] then.

Implementation with Logic. The agent specification can effectively be transformed into running Prolog code. The resulting program consists of several Prolog modules, which reflect the layered specification of the agent. The actual implementation, which also contains an explicit state machine module, is described in [2]. The soccer agents are designed with a three-level approach (Figure 1): the *mode level* contains the most abstract desires an agent has (e.g. setup, attack, defend); the *script level* provides plan skeletons that are used as long as the mode is not changed (e.g. marking, passing, role exchange); the *skill level* hosts basic actions (e.g. kick, dribble).

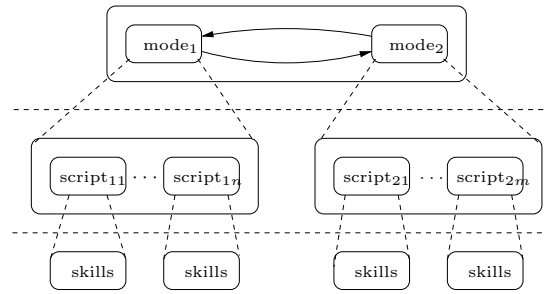

Fig. 1. Three-level approach.

Extension by Utility Functions. The design of adaptive agents with the method presented so far is possible, but it is a good idea to extend the approach by a more adaptive action selection mechanism and to facilitate a more explicit representation of goals of an agent. Until now, the measure for the expected success are Boolean conditions annotated at

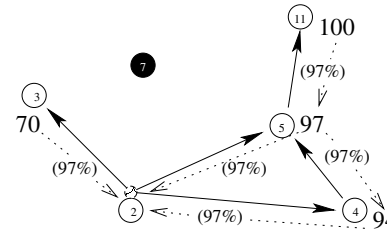

Fig. 2. Searching for pass partners.

transitions. Actions with firing conditions are expected to be successful. If an agent has more than one option to execute actions, the first applicable one is selected. A more adaptive action selection mechanism should evaluate the usefulness of applicable options and execute the most useful one. Therefore, [5] proposes the use of *utility functions* in order to provide a mechanism to evaluate the utility of a script in the current situation taking the commitment to selected options into account.

In our team, utility functions are used for finding the best pass partner. The utility function for pass partners prefers teammates closer to the opponent goal and further away from the own goal. The search for a pass partner is executed recursively for a fixed number of steps, so that a player prefers pass partners who can pass to teammates in a better position. Only in the last step of the search or if no teammate in a better position can be found, the value for the player's position is taken. In other cases, the preference value of the best pass partner is used, so that players with good passing opportunities are preferred. Since we want the players to prefer direct passes to a teammate over a chain of passes to the same teammate, we discount each pass in the preference value of the pass partner (see also Figure 2, where player 2 is about to pass to player 5).

3 Further Features

Ball Interception. An important feature of a soccer agent is *ball interception.* The interception time can be computed effectively. In the soccer server, the ball speed at time t is calculated as $v(t) = \mu \cdot v(t-1)$ with $\mu < 1$. Therefore, if the velocity of the ball is v_0 at time $t = 0$, we have $v(t) = v_0 \cdot \mu^t$. For the distance s, that the ball has moved after t steps, it holds:

$$s(t) = \sum_{i=1}^{t} v_0 \cdot \mu^{i-1} = v_0 \cdot \frac{1-\mu^t}{1-\mu}$$

In the sequel, we assume that an agent can move in any direction with a fixed velocity v_1. At time $t = 0$, the ball is at position $\vec{a}$ in the Cartesian coordinate system with the agent at its origin. Let $\vec{b}$ (with $\|\vec{b}\| = v_0$) be the velocity vector of the ball in this coordinate system. Then, after t steps the position of the ball is $P(t) = \vec{a} + s(t) \cdot \vec{b}$.

Clearly, the agent can reach the ball at any time t with $\|P(t)\| \leq v_1 \cdot t$. Since there is no closed form for t, we apply *Newton's method* in order to find the zeros of $f(t) = \|P(t)\| - v_1 \cdot t$. We compute the following sequence t_n, until $|f(t_n)| < \epsilon$ for some small threshold $\epsilon > 0$:

$$t_n = \begin{cases} 0, & \text{if } n = 0 \\ t_{n-1} - \frac{f(t_{n-1})}{f'(t_{n-1})}, & \text{if } n > 0 \text{ and } f'(t_{n-1}) < 0 \\ 999, & \text{otherwise} \end{cases}$$

This procedure eventually yields the first of at most three zeros $t > 0$. There exists at least one zero; it is found at latest after t_n has been set to 999, which

avoids oscillation. If there are three zeros, then Newton's method will find the smallest one. This follows from the fact that the acceleration a of the ball (the derivative of v) is negatively proportional to v. A similar (but different) method for computing the interception time has been described in [9].

Stable Marriage. Close marking can be very helpful in a variety of situations. By blocking the path between the ball and an opponent, the agent prevents direct passes. But for close marking to work it is important that the agents select the opponents to mark in an intelligent way. Otherwise situations arise in which opponents are marked by more than one agent while others remain completely free. In addition each player should mark an opponent that is as close as possible to avoid stamina consuming dashes across the field.

This mapping of teammates to opponents can be seen as an instance of the *stable marriage problem* [1]. The ranking of the opponents by each player is based on the distance from the player to the individual opponents. It is then possible to generate a solution which assigns the closest possible opponent to each player. If all players have complete knowledge of the positions of teammates and opponents on the field, each opponent will be marked by exactly one player.

Qualitative Reasoning. For the classification of situations occurring during a match, mapping of quantitative numbers to qualitative data is helpful to describe situations in a shorter and more natural way for the programmer. Usage of qualitative predicates for distances and directions is already supported by our C++ interface.

With the *soccer server* version 7, a new application for qualitative reasoning becomes obvious: the *online coach* can define regions, and inform or advise players about the current situation or actions they should perform. These regions can be viewed as a qualitative abstraction from the quantitative data the online coach gets. To effectively use the defined regions, the coach agent has to reason about the implications of its advice to other players. Besides the development and implementation of a coach using qualitative spatial data, qualitative reasoning about velocities and object movement is one of our next planned activities.

References

[1] D. Gale and L. Shapely. College admissions and the stability of marriage. *American Mathematical Monthly*, 1962.

[2] J. Murray. Soccer agents think in UML. Diplomarbeit D 610, Fachbereich Informatik, Universität Koblenz-Landau, 2001.

[3] J. Murray, O. Obst, and F. Stolzenburg. Towards a logical approach for soccer agents engineering. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, LNAI 2019, pages 199–208. Springer, Berlin, Heidelberg, New York, 2001.

[4] Object Management Group, Inc. *OMG Unified Modeling Language Specification*, 1999. Version 1.3, June 1999.

[5] O. Obst. Specifying rational agents with statecharts and utility functions. In *Accepted paper at RoboCup International Symposium (RoboCup 2001)*, 2001. To appear.
[6] A. Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. In T. Ito and A. R. Meyer, editors, *International Conference on Theoretical Aspects of Computer Software*, LNCS 526, pages 244–264, Sendai, Japan, 1991. Springer, Berlin, Heidelberg, New York.
[7] F. Stolzenburg. Reasoning about cognitive robotics systems. In R. Moratz and B. Nebel, editors, *Themenkolloquium Kognitive Robotik und Raumrepräsentation des DFG-Schwerpunktprogramms Raumkognition*, Hamburg, 2001.
[8] F. Stolzenburg, O. Obst, J. Murray, and B. Bremer. Spatial agents implemented in a logical expressible language. In M. Veloso, E. Pagello, and H. Kitano, editors, *RoboCup-99: Robot Soccer WorldCup III*, LNAI 1856, pages 481–494. Springer, Berlin, Heidelberg, New York, 2000.
[9] P. Stone and D. McAllester. An architecture for action selection in robotic soccer. In *Fifth International Conference on Autonomous Agents*, 2001.