RoboLog Koblenz 2002 – Team Description*

Jan Murray, Oliver Obst, and Frieder Stolzenburg

Universität Koblenz-Landau, AI research group
Universitätsstr. 1, D–56070 Koblenz, GERMANY
team@robolog.org · http://www.robolog.org/

1 Background

Outline. For RoboCup agents, a modular architecture for multiagent systems is desirable, that enables us to build a useful *world model* for each agent, that includes information about the position of the agent and other objects, and a history of past situations. It should also be possible to specify high-level behavior, such as a kick to a certain destination and cooperative behavior. Therefore, 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 [8].

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* [7] provide an adequate means. Therefore, the approach of the team *RoboLog Koblenz 2002* employs techniques from software engineering and artificial intelligence research by using UML statecharts and implementing them systematically in *Prolog* [6].

The RoboLog Team. The RoboLog team participated in the simulator competitions in 1999 (Stockholm), 2000 (Melbourne), and 2001 (Seattle). 3 people, Jan Murray, Oliver Obst and Frieder Stolzenburg (team leader), form the core of the team. There are currently 7 additional members, namely Heni Ben Amor, Joschka Bödecker, Marion Levelink, Jana Lind, Christoph Ringelstein, Markus Rollmann, and Karsten Sturm. As in previous years, 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 present the major (new) features of RoboLog agents. The main change of our team is that we re-implemented the computation of the world model, where we distinguish the sensed, calculated, global and fullstate model. We also describe our approach with an explicit state machine. Building agents for a scenario such as the RoboCup also requires the careful and efficient programming of low-level facilities. Robust self-localization, kicking and dribbling are important features. We end with remarks on future work and our work towards coaching and visualization in Section 3.

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

2 Team Features

World Modeling. In the RoboCup scenario it is very important for a soccer agent to have a model of the world which is as complete and precise as possible. As the environment is only partially available to the agent's sensors at any given moment, the world model should not only map the sensor readings to an internal representation but also try to keep track of objects that the agent knows about, but which are currently not picked up by any sensors. The information kept in the world model lay the basis for any decision process the agent may use in order to select an appropriate behavior to execute. But the world model may also aid an agent in certain diagnostic tasks like determining the effects of an action it executes. If the agent has a model of its environment, which enables it to predict a future state of the world based upon current observations, the agent may also determine malfunctions in its sensors or actuators.

The RoboLog 2002 agents are equipped with several world models which are used for capturing different views of the world. The *global world model* is used as the basis for the selection of actions by the agent. In this world model the agent's beliefs about the world and its latest sensor readings are merged to form a consistent model of the world in the current simulation step. Inputs from the various sensors, i.e. see, hear, and sense-body messages are collected in the *sensed world model*. The *calculated world model* is derived from the global world model of the last simulation cycle by means of applying the formulas that constitute the physical model used within the soccer server. Finally, the *fullstate world model* is used for storing the data conveyed by fullstate messages. Since fullstate information are not available during a RoboCup competition, this world model is only used for debugging and evaluation purposes.

At the beginning of each simulation cycle, the world models are updated as follows. Starting from the last global world model the new calculated world model $\mathcal{C}$ is generated. In a second step the new sensor inputs, which have been collected in the sensed world model $\mathcal{S}$, are merged into $\mathcal{C}$ to yield the new global world model $\mathcal{G}$, which is then used by the agent. For diagnostic purposes, a copy $\mathcal{C}'$ of the calculated world model can be made just before the sensor information are included. Then it is easy to compare the predicted state of the world with the actual observations made by the agent, because $\mathcal{C}'$ and $\mathcal{G}$ differ only in those parts that are influenced by sensor inputs. If these discrepancies are too large to be explained by errors or tolerances in the sensor readings, the agent may infer that either one of its components or the world is not functioning as expected. In addition to using different world models, a small history of world models, which covers the last few simulation cycles, is kept by the agent in order to further enhance the assumptions the agent makes about the environment or its own state. See also Figure 1. There, dashed arrows indicate optional steps.

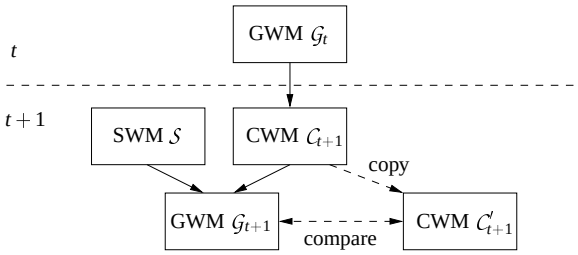

Fig. 1. Updating the global world model.

Structured State Machines. Statecharts are a part of UML [7] and a well accepted means to specify dynamic behavior of software systems. As stated in [4], they can also be used for the design of multiagent systems for the description of plan models for single agents. We use them for the specification of the whole system that consists of more than one agent in general (see also [1, 5]). In statecharts, states are connected via transitions with annotated conditions (subdivided into event and guard) and actions. We distinguish three types of states, which are distinguished by the cardinality of the corresponding sets of initial (sub)states: simple states have 0, composite states have 1, and concurrent states 2 or more *initial states*.

The behavior of agents or their state machines is described by sequences 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 in Prolog 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 then (see [1, 5]). Figure 2 shows the core of our state machine in Prolog.

```
step([State|_],Tree) :-
        trans(State,Next),   % state transition
        !,
        complete(Next,Tree). % complete configuration
step([Top|Sub],[Top|Tree]) :-
        maplist(step,Sub,Tree).
step([],[]).

complete(State,[State|Complete]) :-
        init(State,Init),    % get initial states
        maplist(complete,Init,Complete).
```

Fig. 2. State machine kernel in Prolog.

Further Features. The RoboLog system provides an extensive library that makes precise *object localization* possible. The whole procedure implemented in the RoboLog kernel is able to work even when only little or inconsistent information is given. We (re)implemented the method for mobile robot localization using landmarks stated in [2]. If the corresponding equation system in complex numbers is over-determined, the procedure estimates the position applying the least squares method.

The ability of a soccer agent to perform *hard and precise kicks* is very important. Therefore, a set of kick routines has been implemented which allows us to perform precise kicks to given destinations [3]. Also the velocity can be determined. In order to evaluate the performance of the kick routines that also provide exceptional behavior, we made some tests in fullstate mode.

3 Conclusion

Future Work. With the restricted communication for this year, the need for other means to coordinate the team became more essential. Together with the new rule for kick-ins, it seems to be a good idea to make use of assumptions about the behavior of teammates. This was done by the CMU teams before, where these assumptions were called *locker room agreement* [9].

Coach and Visualization. We have also been programming an online coach in C++ for several weeks now. At the moment we concentrate on *opponent modeling*: finding out which role (attacker, midfield player, or defender) each opponent player has and of which player type the exchanged opponent players are in order to oppose them players of our team with appropriate properties.

Besides taking part in the coach competition for the first time, we also start to participate in the presentation competition. The idea to develop a visualization system arose during a game programming practical. The main goal was to create a visually appealing presentation rather than to simulate real soccer players.

With these different types of competitions, the simulated soccer environment can be used for both our research as well as educational purposes at our university. For our ongoing research, however, it would be helpful to have some kind of long term plan for upcoming changes in the simulated soccer world.

References

1. T. Arai and F. Stolzenburg. Multiagent systems specification by UML statecharts aiming at intelligent manufacturing. In *Proceedings of the 1st International Joint Conference on Autonomous Agents & Multi-Agent Systems*, Bologna, Italy, 2002. To appear.
2. M. Betke and L. Gurvits. Mobile robot localization using landmarks. *IEEE Transactions on Robotics and Automation*, 13(2):251–263, Apr. 1997.
3. J. Bödecker. Der universelle Kick für den RoboCup. Studienarbeit, Fachbereich Informatik, Universität Koblenz-Landau, 2002.
4. D. Kinny and M. Georgeff. Modelling and design of multi-agent systems. In J. P. Müller, M. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III: Agent Theories, Architectures, and Languages*, LNAI 1193, pages 1–20, Budapest, 1997. Springer, Berlin, Heidelberg, New York. Proceedings of Workshop at European Conference of Artificial Intelligence, 1996.
5. J. Murray. Specifying agents with UML in robotic soccer. In *Proceedings of the 1st International Joint Conference on Autonomous Agents & Multi-Agent Systems*, Bologna, Italy, 2002. To appear.
6. 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.
7. Object Management Group, Inc. *OMG Unified Modeling Language Specification*, September 2001. Version 1.4.
8. O. Obst. Specifying rational agents with statecharts and utility functions. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-01: Robot Soccer WorldCup V*. Springer, 2001. To appear.
9. P. Stone. *Layered Learning in Multi-Agent Systems*. PhD thesis, School of Computer Science, Carnegie Mellon University, Dec. 1998.