The NOAI Team Description

Anneli Dahlström, Fredrik Heintz, Martin Jacobsson,
Johan Thapper and Martin Öberg

Department of Computer and Information Science, Linköping university
SE-581 83 Linköping, Sweden
E-mail: {annda, frehe}@ida.liu.se, {marja319, johth341, marob265}@student.liu.se

Abstract. The NOAI team is built on the RoboSoc framework, which is a system for developing RoboCup agents for educational use. The decision making is behavior based and is a continuation of the work of Paul Scerri et al. on the Headless Chickens III. We have extended their decision trees to directed acyclic decision graphs and made it possible to use full predicate expressions instead of just predicates. We have also added a language for describing the decision graphs. This is to be able to use their, highly successful, strategy editor to graphically design the team strategy.

1 Introduction

This paper describes the most important aspects of the RoboCup simulation team NOAI. The team is built on the framework provided by RoboSoc [2], which is a system for developing RoboCup agents developed for educational use. The team is a part of the evaluation of RoboSoc.

Since we want to use the strategy editor developed by Johan Ydrén and Paul Scerri [4] we use a behavior based decision making that closely resembles their approach. The strategy editor was one of the most important factors in the success of Headless Chickens III (HCIII) in RoboCup'99 in Stockholm.

2 RoboSoc

The purpose of RoboSoc is to simplify the development of RoboCup agents by taking care of the basic problems that an agent has to handle, like interacting with the soccer server and keeping track of the current simulation cycle.

The resulting system consists of three parts, the library, the basic system and the framework. The library consists of basic objects and utilities used by the rest of the system. The basic system takes care of the interactions with the server, like sending and receiving data. It is also responsible for the timing and most of the support for decision making, since it generates events for the decision maker to react on when the world or the internal state of the agent changes.

The framework defines three concepts, used for world modeling and decision support, *views*, *predicates* and *skills*. The views are specialized information processing units responsible for a specific part of the world model, like modeling

P. Stone, T. Balch, and G. Kraetzschmar (Eds.): RoboCup 2000, LNAI 2019, pp. 413-416, 2001.

the ball or the agent. They are also controlled by events generated by the basic system. The predicates can be used either by the decision maker or the skills to test the state of the world. They work like predicates in a fuzzy logic, and can have a value between zero and one depending on the certainty of the answer. The skills are specialized, short-term planners which generate plans for what actions the agent should take in order to reach a desired goal state. For more information see [2].

3 High Level Decision Making

The decision making architecture is based on the work of HCIII [3]. It uses the same two layers as they do, one for low level skills and one for high level strategy. The strategy layer access the information in the world model through a given set of fuzzy predicates. It also use a behavior based decision making, similar to HCIII, but with some modifications to achieve more efficiency, flexibility and expressiveness. The architecture of the behavior hierarchy tree is described in [1].

In the following text we will separate the person doing the programming of the agent from the person designing the behavior of the agent. The reason is that the agent behavior designer could (or maybe should) be a domain expert, in this case a soccer coach or similar, without programming skills. When we refer to the programmer we mean a person that is capable of changing the actual code and then recompile the program, while the agent designer only needs to change configuration files that are loaded when the agent is started.

3.1 Behavior DAG

The task of a behavior is to choose a lower level behavior to activate. Therefore every behavior is defined by a list of lower level behaviors and the technique used to choose among them. The choice is made with the concept of activations. Every behavior has an activation level and the behavior with the highest activation level is chosen. The activation level is increased or decreased every cycle based on the value of a predicate expression. The amount of change can be specified for each behavior. Different behaviors uses different predicate expressions.

In HCIII the behaviors are layered, a behavior in layer n can only activate behaviors in layer $n-1$. The layer 0 only contains skills, which sends the actual server command to the soccer server when they are activated.

The layering is unnecessary, it can be generalized to let the behaviors form a rooted directed acyclic graph (DAG) where the nodes are behaviors and the terminals are skills. The decision is made by finding a way from the root behavior through the DAG to a skill and activating it.

The agent designer, who constructs the behavior DAG, may of course still use a layered approach but it is not necessary. She can for instance layer the behavior DAG as in Figure 1.

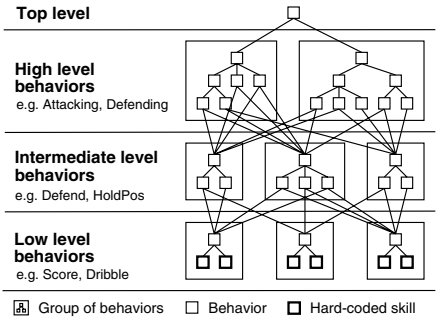

Fig. 1. Example layering and grouping of behaviors.

3.2 Predicate Expressions

The interface between the RoboSoc view system and the behavior based decision system is the predicates. The predicates are programmer defined objects that use the information stored in the views and return a fuzzy boolean value. The predicates hides details so the agent designer can use high level information for the decision making. The predicates are used to update the activation levels in the behaviors.

Since the predicates are predefined by the programmer, the agent designer can not create new predicates when needed. To try to overcome this shortcoming, the designer can create fuzzy logic formulas of predicates with logical connectives. The formulas can be used when updating an activation instead of a single predicate, which was the case in HCIII. Especially the logical connective `not` is very useful, but `and`, `or` and `xor` are also available.

3.3 Activations

The concept of activations in HCIII is very useful, it can make the behavior of the agent very reactive and at the same time delay changes in actions if necessary. By delaying changes, the agent can handle temporarily incorrect information and avoid oscillating between behaviors.

One problem is comparing two options. For example if an agent chooses between dribble to the left or to the right. The decision should be based on the number of opponents to the left and to the right. You want to compare the number of the opponents in these areas.

This is solved by constructing another kind of predicates. Predicates that do not answer a boolean question, but instead gives a gauge value. One example is the NumberOfOpponentsInArea, which returns zero if no opponents are in the area and one if all 11 opponents are there. Now you need a special type of behavior that does not use activations when choosing the behavior to activate. This second type of behavior will still have a predicate associated with each choice, but instead of updating activations and choose the option with the highest

activation level, it simply choose the behavior with the highest predicate value. It is up to the agent designer to decide which behaviors should use activations.

A second advantage with these behaviors without activations is efficiency. Every activation in the behavior DAG must be updated every cycle by calling its predicates, even if that part of the graph is never activated. With the behaviors without activation only the active part of the graph needs to call its predicates. By using as many behaviors without activation as possible we can improve the overall efficiency of the agent and possibly have a larger behavior DAG.

3.4 Behavior Description Language

The behavior DAG is specified in a text file which means that you do not need to recompile your agent to change the behavior of the agent. When the agent is started it reads the text file and creates the behavior DAG. The language used to describe the behavior DAGs is made of S-expressions, that is a LISP-like syntax. The files can be created in any text editor, but also by tools such as a strategy editor [4]. Since a language for specifying the behaviors is defined, you can build different types of tools for creating behaviors.

4 Future Work

One possible direction for future work is to extend the behavior system and try to implement as much as possible of the skills with behaviors. Instead of having a dribble skill, it could be possible to build a dribble behavior by using simpler skills like, MoveToXY, KickBallSoftlyToXY, KeepBallToYourLeft and so on. Then more functionality can be decided by the agent designer instead of the agent programmer.

The language for defining behaviors makes it possible to use different kinds of tools to create behaviors. It would be interesting to compare different tools, to see how the speed of development and quality of the soccer agents depend on the different tools. It might also be interesting to apply machine learning in the process of creating strategies and behaviors.

References

[1] Silvia Coradeschi, Paul Scerri, and Anders Törne. A User Oriented System for Developing Behavior Based Agents. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*, pages 173–186. Springer Verlag, Berlin, 1999.

[2] Fredrik Heintz. RoboSoc a System for Developing RoboCup Agents for Educational Use. Master's thesis, IDA 00/26, Linköping university, Sweden, March 2000.

[3] Paul Scerri, Johan Ydrén, Tobias Wiren, Mikael Lönneberg, and Per-Erik Nilsson. Headless Chickens III. In Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*. Springer Verlag, Berlin, 2000.

[4] Johan Ydrén and Paul Scerri. An Editor for User Friendly Strategy Creation. In *Robocup 1999 Team Description: Simulation League*. Linköping Electronic Press, 1999.