End User Specification of RoboCup Teams

Paul Scerri and Johan Ydrén
Real-time Systems Laboratory
Department of Computer and Information Science
Linköpings Universitet, S-581 83 Linköping, Sweden
pausc,x98johyd@ida.liu.se

Institute for Computer Science, Linkpings Universitet

Abstract. Creating complex agents for simulation environments has long been the exclusive realm of AI experts. However it is far more desirable that experts in the particular application domain, rather than AI experts, are empowered to specify agent behavior. In this paper an approach is presented that allows domain experts to specify the high-level team strategies of agents for RoboCup. The domain experts' specifications are compiled into behavior based agents.
The 1999 RoboCup World Cup provided an interesting basis for evaluation of the approach. We found that for RoboCup it is not necessary to allow a user to change low level aspects of the agents' behavior in order for them to create a range of different, interesting teams. We also found that the modular nature of behavior based architectures make them an ideal target architecture for compiling enduser specifications.

1 Introduction

Simulation environments where intelligent agents play the roles of humans are used for training and testing in domains such as air combat training, fire fighting command training and large scale military simulations. The development of intelligent agents for such environments has been, and continues to be, a challenge for AI researchers. However over the years a wide range of techniques have been developed to meet many of the challenges faced. Nearly all of the developed techniques rely on the availability of agent experts to program the agents. This is highly undesirable in cases where there is either a large body of expert knowledge that needs to be incorporated into the agent or when the agent behavior needs to be changed often. Either case implies a large amount of work, better off done by a domain expert, being done by an agent expert. In this paper an approach is presented aimed at bridging the gap between domain experts and agent programming for one particular domain, namely RoboCup[3].

Our approach is to provide a graphical editor, resembling a coach's whiteboard, with which a domain expert can specify the high level strategies of a team in a manner similar to the way he might explain strategies to a real soccer team. For example, Figure 1 shows a diagram explaining one professional soccer strategy taken from the Internet. The diagram seems to be typical of the way

complex, high level strategies are explained by humans for humans. The strategy editor presented here attempts to mimic this explanation style.

In our system a strategy is specified by drawing circles for player positions and arrows for the directions to pass and dribble the ball. The whiteboard style editor allows a wide range of options and flexibility providing the ability to specify complex strategies without knowledge of the underlying agent architecture. The specification, done at a team level, is subsequently compiled into eleven separate behavior-based agents, one for each player.

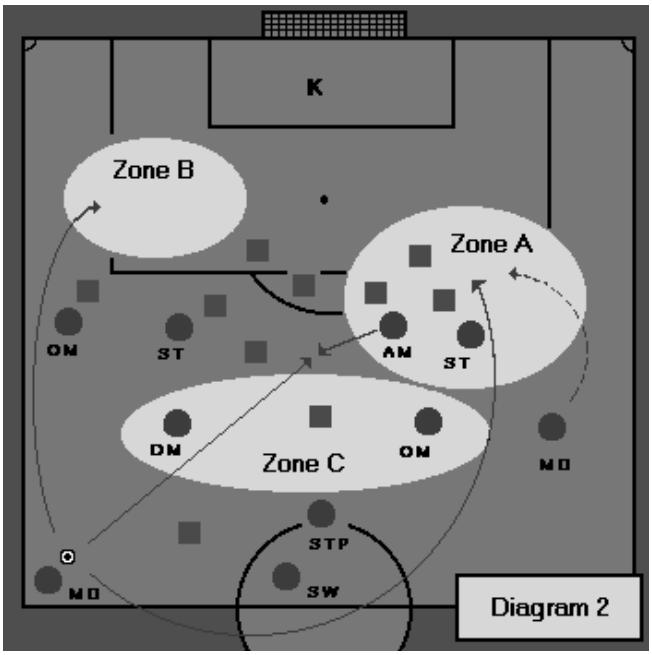

Fig. 1. A diagram from the Internet explaining a particular team soccer strategy to a human audience.

Although the system described here is intimately tied to the RoboCup domain it is hoped that lessons learned from the system can be applied generally to the problem of empowering domain experts to program agents for complex, agent-populated simulations. In particular an evaluation of the system provides insight into two questions. Firstly, are behavior based architectures good target architectures for compiling end user behavior specifications? Secondly is it sufficient in some domains to give endusers *only* the ability to specify the high level behavior of the agents?

2 The Editor

The heart of this approach to domain expert specification is a flexible, visual editing tool called the *strategy editor*. A soccer expert (henceforth referred to as the designer) indicates positions of players and intended ball movement in a style similar to drawing on a whiteboard. To create a particular team strategy the designer places all players on an image of the playing field. For each player the designer can specify an arbitrary number of positions to pass to, an arbitrary number of directions in which to dribble and directions to watch for incoming passes. The compiled agent will use its knowledge of the world (e.g. relative positions of other players) to choose at runtime which option to take when it gets the ball. Because all the players are simultaneously shown in the editor, and the designer is working on them all concurrently, it is trivial to ensure that players pass to positions where team mates are likely to be (and correspondingly that players move to positions where passes are likely to come). Soccer experts can place players and design passing and dribbling formations that lead to good RoboCup teams. Notice that once players are compiled any team behavior that occurs is "emergent", i.e. there is no explicit communication or explicit representation of passing patterns. In effect the strategy editor provides a means to visualize and specify "emergent" team behavior.

As well as control over *what* the players do, i.e. where they pass and dribble, a designer has some control over *how* the player does something. For each player the designer may choose the *style* with which the player will play (the available styles are determined from the player skeleton – see below). Some general styles are *normal play* or *shooter*, while more specialized styles are *crosser* (always tries to kick the ball to the middle) and *wait* (just watches ball, useful when referee stops play, etc.) The styles influence how the player fulfills its part of the designer's strategy. For example, the shooter style results in a player that will chase the ball in a fairly large area around its assigned position and will shoot if at all reasonable. Apart from selecting styles of play the designer has no low level control over the behavior of the players. The details of the players' behavior come from a template created in a low level individual player strategy editor.

The process of designing formations and selecting styles is repeated for each of the different game "modes" (or situations) that the player will distinguish between. Example modes are *kickoff*, *deep defense* and *transition to attack*. The modes of play that the player knows about are determined from a player "template" created in a lower level individual strategy creation tool (see below and [4]). For RoboCup99 there were about 15 different modes. At runtime the player determines the current mode mainly by the position of the ball on the ground but also considers factors such as referee calls and which team is nearer the ball. It is possible to view the strategies more than one mode at once, hence it is possible to see how a player must move when the team switches from offense to defense or vice versa.

Figure 2 shows a designer specification of how a particular defensive strategy should work. Circles represent players. Arrows with double lines indicate the direction that the player should dribble when it has the ball. Arrows with single

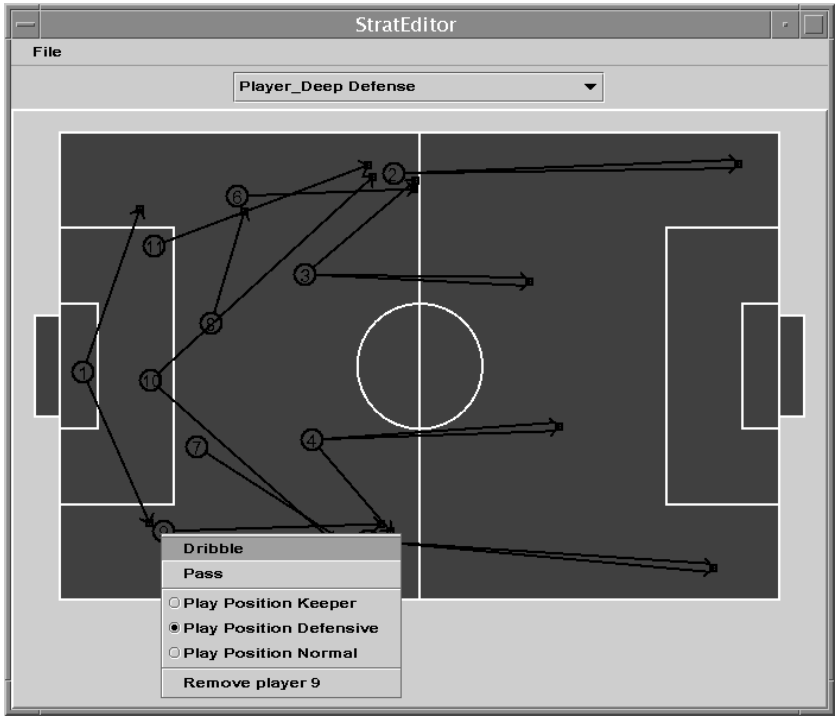

Fig. 2. A diagram of a team's defensive strategy. A player is represented by a circle. An arrow with a double line represents a direction to dribble. An arrow with a single line represents a direction to pass.

lines indicate where the player should pass when it has the ball. Priorities are not shown on this diagram. The main idea behind this particular strategy is that from defense the ball should be cleared via the sides of the ground. Notice that many of the players have multiple options when they have the ball. At runtime the agent will choose the most appropriate action that is applicable, e.g. when the player has the ball and there are opponents close the action chosen will be the pass that a teammate is most likely to receive. Notice also how the pass directions for one player are easily specified to match the positions where other teammates are likely to be situated.

3 Compiling Team Specifications to Behavior Based Trees

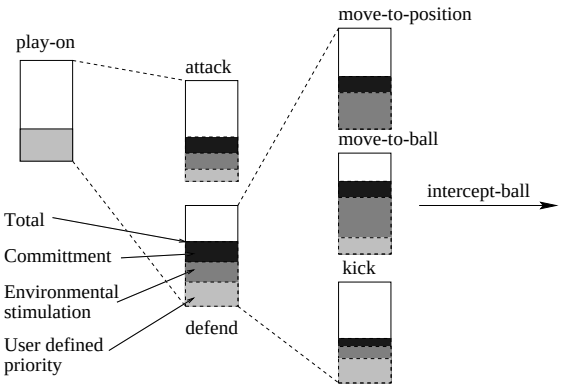

Fig. 3. A snap shot of an agent's reasoning. The grey boxes show the contributions of different components of the activation function to the total activation of the behavior.

The strategy editor is "configured" with a "skeleton" agent. The skeleton is a hierarchy of parameterized behaviors that can be subsequently instantiated into an agent. The skeleton agent can be thought of as a generic player. Three keywords are embedded in the specification of the generic player: MODE; ATTR (attribute) and ACTION. The team strategy editor uses the template (especially the keywords) to configure itself, i.e. by creating a panel for specifying each MODE, a menu item for each style (i.e. ATTR) in the mode and the ACTIONS the player can take with the style. This mechanism makes the strategy editor very flexible, quite different skeletons can be used to create teams with quite different abilities.

Once a team has been specified, partially or completely, it is compiled into a layered behavior-based agent. The process of compiling a team strategy specifica-

tion consists of instantiating parameters and groups of behaviors in the skeleton hierarchy. A separate skeleton is instantiated for each player.

The architecture of the resulting agents is described in more detail in [4]. An agent is composed of a hierarchy of behaviors. At each level of the hierarchy a single behavior is chosen to act. The behavior chosen is the one with the highest activation where the activation is a function of the behavior's priority, commitment and the prevailing environmental conditions. Behaviors at higher levels of the hierarchy act by setting appropriate lower level behaviors and the lowest level behaviors act by turning on or off simple skills. Figure 3 shows a snapshot of part of a behavior hierarchy, indicating the way that activation is calculated and how it effects which behavior is selected to act.

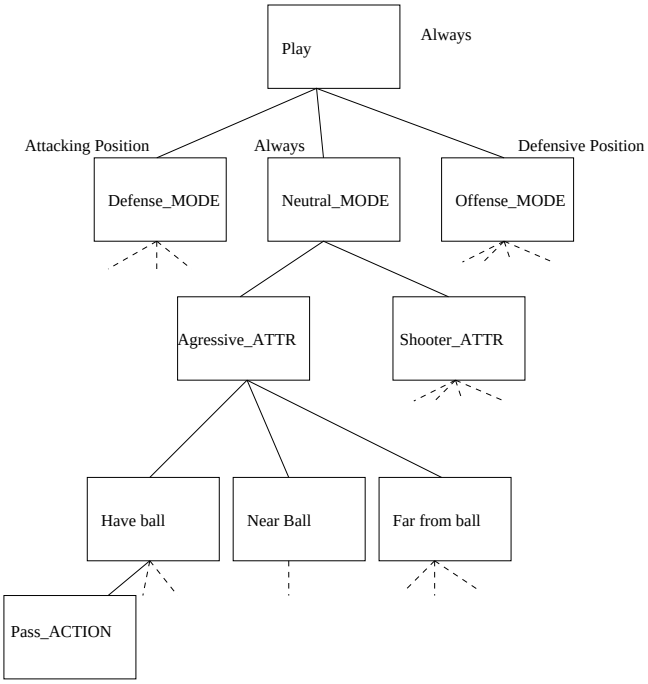

Fig. 4. Parts of the generic player's behavior hierarchy. Boxes represent behaviors. Each behavior is labeled with its name. Next to each box is the activation condition associated with the behavior.

Each team formation, i.e. one for each game mode, corresponds to one group (usually a hierarchy) of behaviors in the agent. Figure 4 shows the top part of an agents hierarchy of behaviors. Notice that there is one branch for each game mode (indicated by the keyword MODE). Appropriate conditions are associated with each hierarchy to ensure that the correct hierarchy is activated at the correct

time in the game, for example the defensive strategy will be activated when the ball is in the back third of the field. A designer need not know the condition associated with the strategy.

Depending on the style chosen for the player a slightly different generic tree (indicated by keyword ATTR) is selected (by the compiler) for the agent. The hierarchies vary only slightly, mainly in terms of the functions for activating different behaviors. For example in a cautious style hierarchy behaviors for checking the location of the ball are more readily activated.

The details of the specified strategies are then used to instantiate the details of the style hierarchies. The specified position of a player on the field in a strategy is directly translated into parameters for a corresponding behavior that moves the player to a position. Figure 5 shows how one specified position has been instantiated to parameters in a behavior hierarchy. Each pass or dribble action specified for a player results in a dedicated behavior (or possibly behavior hierarchy – indicated with the keyword ACTION in the player skeleton) being added to the overall agent hierarchy. Figure 6 shows the resulting instantiated tree after behaviors for two different pass behaviors and a single dribble behavior have been added.

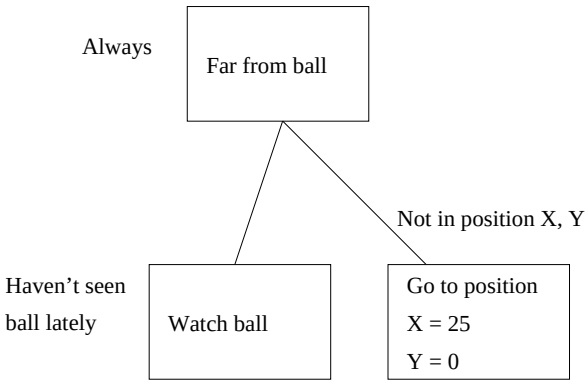

Fig. 5. This diagram shows how the position parameters of a player are incorporated into the tree. The X and Y values for the behavior and for the activation condition are taken directly from the specification.

4 Discussion

A number of different approaches have been, and continue to be, taken to meet the challenge of empowering end users to program. An evaluation of this prototype serves as an evaluation of two ideas. The first idea is that behavior based systems serve as a good target architecture for compiling end user specifications.

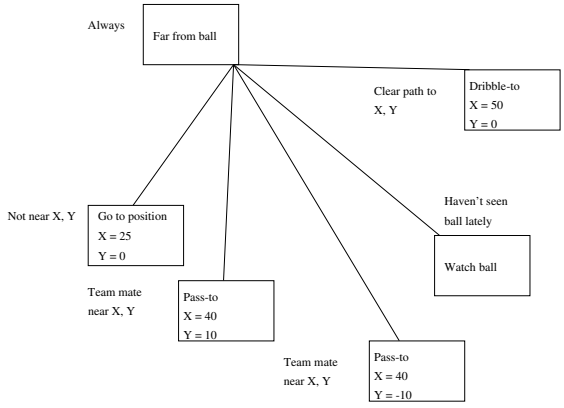

Fig. 6. This diagram shows how the behaviors have been added for two user specified pass directions and one user defined dribble direction.

The second idea is that for this domain, which we do not believe is unique, it is sufficient to give users only the ability to change the high level behavior of agents. The system described here tests both these ideas.

The first idea, that behavior based systems are an appropriate target architecture for end user programming is supported by the intuitive idea that end users describe behavior in the same way as behavior is implemented in a behavior based system. Loosely speaking, users seem to think about behavior as concurrent tasks that one switches between depending on environmental conditions. For example a soccer expert may talk about defense and attack, or dribbling and passing. In a behavior based system each of the described behaviors would be implemented as a separate logical unit (i.e. a behavior). An end user also assumes that other tasks such as maintaining situational awareness and continually evaluating whether the current behavior is most appropriate are done continuously in parallel with the current task – in a behavior based system this is standard functionality (i.e. by switching between low level behaviors).

Because the specification method of the editor is close to the usual explanation method of the designer the "translation" required by the designer is minimal. Any act of communication, even from human to human, results in some loss of information or ambiguity. The more difficult the task of communication the more the information loss and ambiguity. Hopefully by minimizing the translation required the information loss can be minimized. As a contrast consider having a user describe soccer behavior in terms of, say, hierarchical plans or beliefs and desires. Clearly an average user would need to do considerable translation from their normal way of describing what a soccer player does to express his ideas in such a formalism. Surely, it follows, that a large amount of expert knowledge would be lost in translation.

The second idea tested by our system, that a successful end user programming environment need not give programmers low level control over the behavior of agents, is perhaps less contentious. Regardless of the particular high level strategy the basic behavior of the agents does not change substantially, e.g. they still dribble, pass, run etc. in much the same way. For this reason it does not seem essential that the strategy designer be given low level control. In other domains, such as mobile robotics, the low level behavior of the agent may change dramatically with changing high level tactics rendering a high level only approach inapplicable. However simulated soccer is not the only domain where different high level strategies use the same low level behaviors, computer games and air-combat simulation being two other examples.

Unlike in domains such as manufacturing, where design, implementation and testing time may be measured in tens of man years, it is desirable that the design and implementation time of RoboCup strategies be measured in hours if not minutes. In order to support development times of this order of magnitude, the user cannot be required to make too many decisions. Even considering only high level strategies a virtually infinite design space exists. Considering that reasonable design times are unlikely to be achieved if users delve into low level behavior specifications of agents and also that at a high level there are a large range of options it seems reasonable that an end user programming system not allow the user to change the low level behavior of the agents.

During the RoboCup World Cup competition undergraduate computer science students, relatively inexperienced with agent technology, specified the behavior of HCIII and changed it (sometimes markedly) for nearly every game. Although the students became relatively experienced with the strategy editor they made very few accommodations for aspects of the underlying agent behavior and felt as though the players they specified would do as they intended. The players performed as the students intended indicating the transformation from team specification to individual agents was faithful to the designers intentions. This supported our hypothesis that behavior based architectures are a good target architecture for compiling high level specifications. The students developed a surprising range of different team strategies tailored to different oppositions. The most critical result was that the agent template rarely needed to be modified, even though the team behavior varied greatly. The students seemed more than happy to work within the confines of the team editor and managed to produce a wide range of team well tailored to particular opponents. That fact supported the hypothesis that end users did not need to specify low level aspects of the agents in order to achieve results they were looking for.

During the competition itself there was very little time for modifications of teams. However even in the shortest periods (as little as five minutes!) different team strategies, tailored to the upcoming opposition, were developed. In fact the World Cup provided a fantastic opportunity to assess the rapid development capabilities of the strategy editor and the results supported the claim that very rapid, yet effective development was possible.

5 Conclusion

In this paper we described a high level, end user programming environment for specifying team level strategies for RoboCup. The system allows a soccer expert to specify team positions and ball movement in a manner analogous to drawing on a whiteboard. The strategy editor was designed to evaluate two hypotheses, firstly that behavior based architectures were a good target architecture for compilation of high level strategies and secondly, that end users needed only high level specification capabilities when specifying complex teams. The system was put through rigorous "real-life" testing during the RoboCup World Cup. Both hypotheses were supported by the experience. Students, relatively inexperienced with agent technology, specified strategies very quickly and the resulting teams executed the strategies in accordance with the students intentions. This experience leads us to believe that the idea of high level specification by a domain expert followed by compilation into behavior based agents is worth considering in other domains.

Acknowledgments

This work is supported by Saab AB, Operational Analysis division, The Swedish National Board for Industrial and Technical Development (NUTEK), under grants IK1P-97-09677 and IK1P-98-06280, and Linköping University's Center for Industrial Information Technology (CENIIT), under grant 98.6.

References

1. Bruce Blumberg and Tinsley Galyean. Multi-level control of autonomous animated creatures for real-time virtual environments. In *Siggraph '95 Proceedings*, pages 295–304, New York, 1995. ACM Press.
2. Rodney Brooks. Intelligence without reason. In *Proceedings 12th International Joint Conference on AI*, pages 569–595, Sydney, Australia, 1991.
3. Hiraoki Kitano, Minoru Asada, Yasuo Kuniyoshi, and et. al. RoboCup: A challenge problem for AI. *AI Magazine*, 18(1):73–85, Spring 1997.
4. Paul Scerri, Silvia Coradeschi, and Anders Törne. A user oriented system for developing behavior based agents. In *Proceedings of RoboCup'98*, Paris, 1998.