End User Specification for Simulation of Teams of Soccer Playing Robots

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

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 able to specify agent behavior. In this paper an approach is presented that allows domain experts to specify high-level team strategies of agents for an enviroment where simulated robots play soccer.

1 Introduction

Simulation environments, where intelligent agents take the place 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 presented by the domain. Nearly all of the techniques developed 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 as either condition implies a large amount of work by agent experts. In this paper an approach is presented aimed at bridging the gap between domain experts and agent programming for one particular domain, namely RoboCup[Kitano *et al.*, 1997], an enviroment where simulated robots compete in soccer.

Our approach is to provide a graphical editor, resembling a coaches 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 his team before a game. 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.

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 the domain expert having 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.

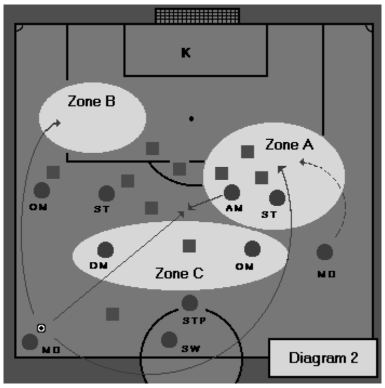

Figure 1: A diagram taken explaining a particular team strategy for other humans.

Although the system described here is intimately tied to the RoboCup domain we hope 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 is hoped to provide insight into two areas. Firstly, are behavior based architectures good target architectures for compiling end user behavior specifications? Secondly is it sufficient in some domains to give users only high level control over agents?

2 The Editor

The heart of the approach 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. The high level strategy is later compiled into a separate controll system for each player. 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 and an arbitrary number of directions in which to dribble. The compiled agent will use its knowledge of the world to choose at runtime which option to take when it receives 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. It is hoped that soccer experts will be able to place players and design passing and dribbling formations that lead to good RoboCup teams.

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. At present the style can be either aggressive, cautious or neutral. The styles influence how the player fulfils its part of the designer's strategy. For example, the cautious style results in a player slightly more careful to keep track of the position of the ball and be more inclined to pass than dribble when opposition players are nearby. 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 are hard-coded into the compilation process that translates the strategy into executable players.

The designer is able to specify different strategies for different modes of play. Currently the editor supports modes for the balls being in an offensive, defensive or neutral position, however this is expected to expand greatly in the future. At runtime the player determines the current mode by the position of the ball on the ground (the condition determing the current mode may become more complex when more modes are added). It is possible to view the strategies in the editor for all modes 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 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 effectively choose the highest priority action which is applicable, e.g when the player has the ball and there are opponents close the action chosen will be the pass which 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

Once a team has been specified, partially or completely, it is compiled into a layered behavior based agent, which controls the agents behavior. The target architecture is described in [Scerri *et al.*, 1998]. Basically 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, committment 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 allowed to act.

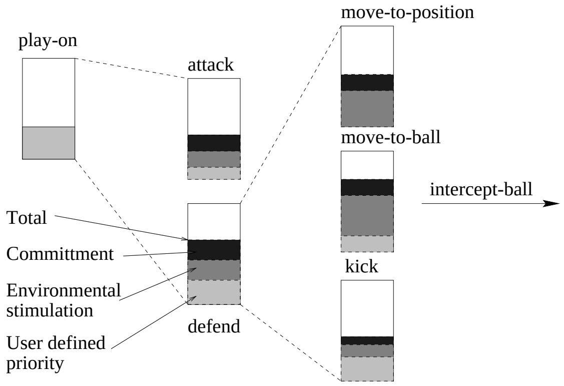

Figure 3: A snap shot of an agent's reasoning. The greyed boxes show the contributions of different components of the activation function to the total activation of the behavior.

Concurrently with the design of the editor a generic skeleton agent was created. The process of compiling a team strategy specification consists of instantiating parameters and groups of behaviors in the skeleton hierarchy. A separate skeleton is instantiated for each player.

Each strategy, 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. 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 team designer need not know the condition associated with the strategy. At a later point the conditions may be made more complex, for example by including the positions of players across the field.

Depending on the style chosen for the player, i.e. aggressive, cautious or neutral, a slightly different skeletal tree is used for a hierarchy of behaviors. The hierarchies

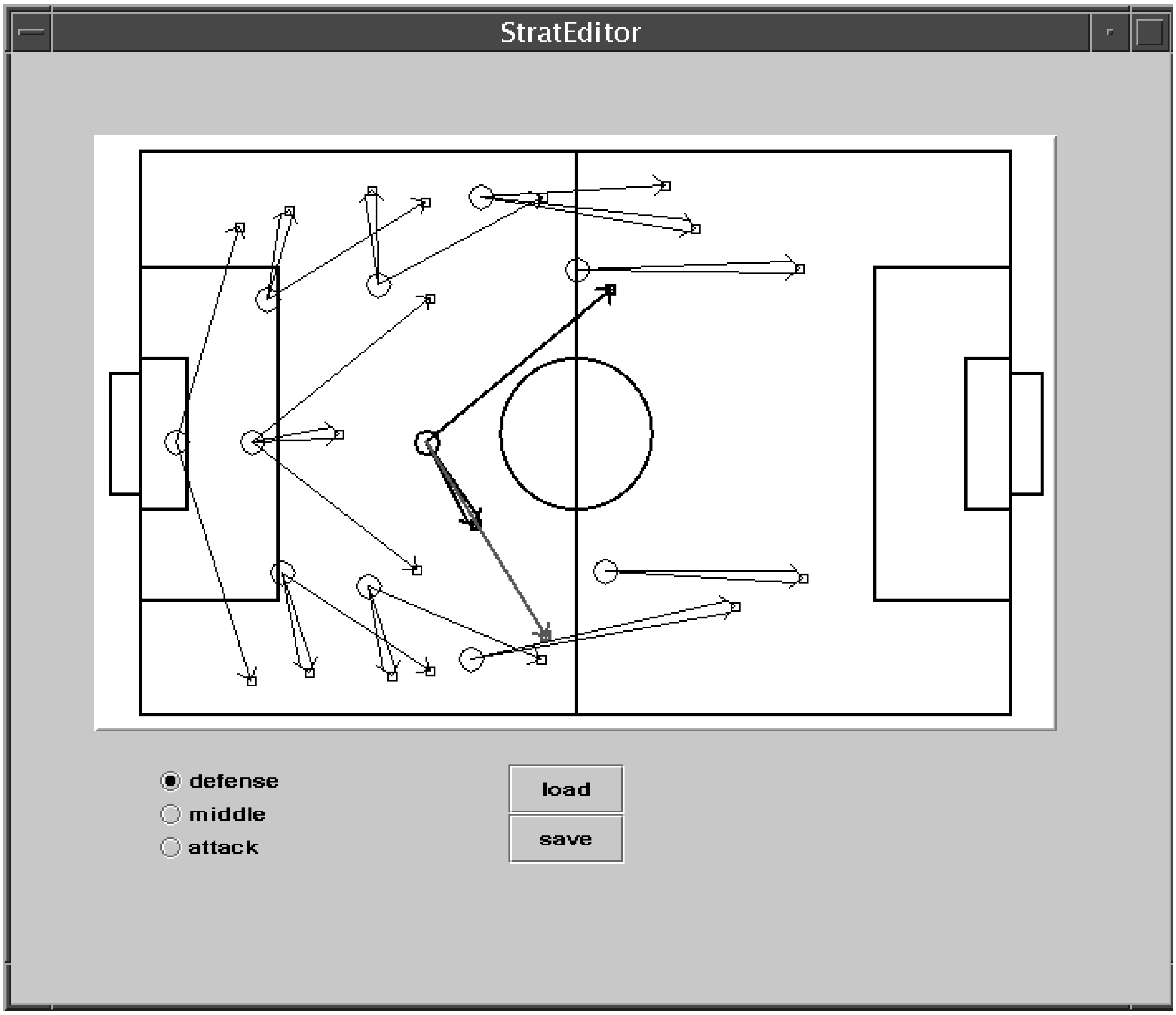

Figure 2: A diagram of the team's defensive strategy. A player is represented by a circle. An arrow with a double line represents a direction to dribble. An arrow with a single line represents a direction to pass.

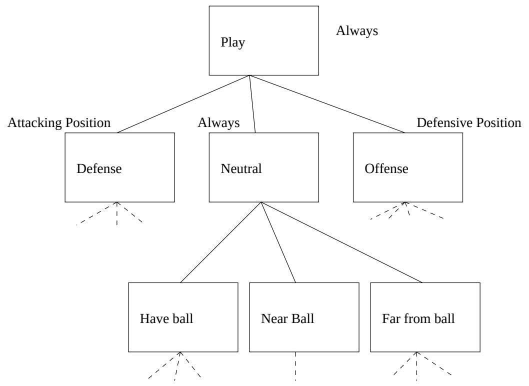

Figure 4: Parts of the generic player's behavior hierarchy. Boxes represent behaviors. Each behavior is labeled with its name. Next to each box is the activation condition associated with the behavior.

vary only slightly, mainly in terms of the functions for activating different behaviors. For example in an agent with a cautious style behaviors for checking the location of the ball are more readily activated.

The details of the specified strategies are then used to instantiate 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) being added to the overall agent hierarchy. Figure 6 shows the resulting instantiated tree after behaviors for two different passes and a single dribble direction have been added. The hierarchy that is added to the agent for a particular action is an instantiation of a generic, hardcoded behavior for that action. The behavior will be appropriately parameterized with information about where its corresponding action should take place, e.g. where to pass to. The activation functions for the behaviors are modified to reflect the priority information specified by the designer.

4 Discussion

A number of different approaches have been, and continue to be, taken to the challenge of empowering end users to program. An evaluation of this prototype can serve as an evaluation of two ideas. The first idea is that behavior based systems serve as a good target architecture for end user programming. The second idea is that for this domain, which we do not believe is unique, it is sufficient to just give users the ability to change high level behavior of agents. We believe useful behavior based agents can be constructed from high level user specifications. The system described here embraces both these ideas.

The first idea, that behavior based systems are an

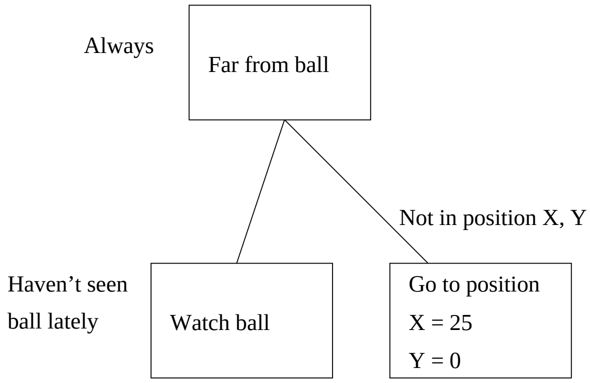

Figure 5: This diagram shows how the position parameters of a player are incorporated into the tree. The X and Y values for the behavior and for the activation condition are taken directly from the specification.

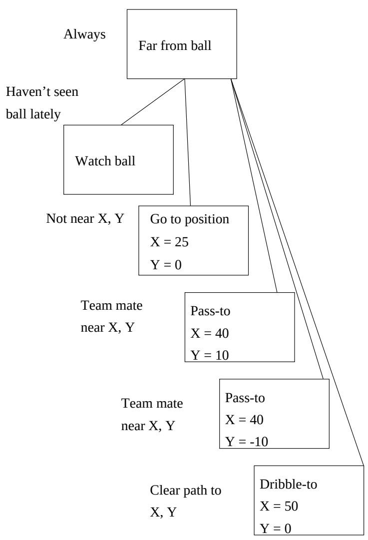

Figure 6: This diagram shows how the behaviors have been added for two user specified pass directions and one user defined dribble direction.

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.

Because the method of specification in the editor is close to the thinking 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.

We believe that the close correspondence between the target architecture and a soccer experts' explanations will make the process of translating end user specifications into executable agents relatively easy. The more direct the translations the easier it is to have the resulting behavior faithfully capture the end user ideas. The more faithfully the system reflects the designers ideas the less the designer needs to worry about specifying things in a way the system will understand and the more they can concentrate on designing appropriate strategies.

The second idea embraced 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 end user to change the low level behavior of the agents.

There appears to have been little other work done on domain expert specification of complex agents. KidSim takes a very different approach to our approach, namely it gives only low level control to a user[Smith *et al.*, 1997]. Behavior is specified by demonstrating the required behavior. The KidSim approach seems effective at giving children the ability to create simple simulations but is unlikely to scale up to more complex domains.Programming by example systems "watch" the behavior of an expert then attempt to copy the behavior, generalizing when possible[Lieberman, 1993]. For some domains programming by example is the holy grail of end user programming because it removes all need for users to perform the difficult task of describing what they want the agent to do in a structured way. However programming by example may not be appropriate in domains, such as RoboCup, where the breadth of possible agent behavior and the (relatively) large number of agents makes it inefficient to specify the whole spectrum of behavior by demonstrating, usually repeatedly, the required behavior. Repenning, with his AgentSheets system[Repenning, 1993], takes a contrasting view that users need access to all aspects of agent development, although low level specification facilities are hidden until high level specification facilities have been mastered. Non-computer expert users of AgentSheets have a wide variety of control over agents, however as with KidSim it does not seem as though AgentSheets agents will scale up to the complexity of the RoboCup domain.

5 Conclusion

In this paper we describe 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. It is hoped that use of this system will help evaluate two hypotheses: low level specification of agent behavior by a strategy designer is not required for some domains; and that behavior based systems are a good target architecture for end user programming. It is hoped that eventually soccer experts who are not computer experts will assist in evaluating the system. Results of such an evaluation will be used to indicate further, necessary features of such a system.

References

[Blumberg and Galyean, 1995] 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.

[Brooks, 1991] Rodney Brooks. Intelligence without reason. In *Proceedings 12th International Joint Conference on AI*, pages 569–595, Sydney, Australia, 1991.

[Kitano *et al.*, 1997] Hiraoki Kitano, Minoru Asada, Yasuo Kuniyoshi, and et. al. RoboCup: A challenge problem for AI. *AI Magazine*, 18(1):73–85, Spring 1997.

[Lieberman, 1993] Henry Lieberman. *Watch what I do: Programming by demonstration*, chapter Tinker: A Programming by Demonstration System for Beginning Programmers. MIT Press, 1993.

[Repenning, 1993] Alexander Repenning. *AGENTSHEETS: A tool for building domain-oriented dynamic, visual environments.* PhD thesis, University of Colorado, Boulder, 1993.

[Scerri *et al.*, 1998] Paul Scerri, Silvia Coradeschi, and Anders Törne. A user oriented system for developing behavior based agents. In *Proceedings of RoboCup'98*, Paris, 1998.

[Smith *et al.*, 1997] David Smith, Allen Cypher, Jim Spohrer, Apple Labs, and Apple Computer. *Software Agents*, chapter KidSim: Programming Agents without a Programming Language. AAAI Press/The MIT Press, 1997.