Agent Roles in RoboCup Teams

Helena Åberg
Kungliga Tekniska Högskolan
School of Computer Science and Technology
February 1998

Abstract

A multi-agent system can sometimes benefit from assigning distinct roles to the agents. The agents will thus have different behaviours and responsibilities. I discuss roles in multi-agent systems and investigate how roles can be used in a RoboCup team of soccer-playing agents. The basis of a generic RoboCup agent has been implemented from which a team of agent players with specialised roles can be constructed.

This thesis corresponds to the effort of 20 full-time working weeks.

Acknowledgements

The work described in this paper was carried out at the Department of Computer and Systems Sciences at the Royal Institute of Technology, Stockholm.

I have been working very closely with my fellow student Åsa Åhman. She has also been writing her thesis but focused on decision control in RoboCup teams. Most of the design and implementation described in this thesis is the joint work of Åsa Åhman and myself.

I would like to thank my supervisor Dr. Magnus Boman for his ideas, constructive criticism, and moral support. I would also like to thank my co-worker Åsa Åhman without whose help the basic and advanced skills of the agents would have been very difficult to implement.

1 Introduction

1.1 Purpose

The purpose of this thesis and the implementation of a RoboCup team is to give some insight to which ideas about the use of roles in agent technology that are useful and practical. This knowledge can be used to build improved and more successful multi-agent systems operating in dynamic environments.

1.2 Methodology

We have partly implemented a RoboCup team with the intention of investigating two areas:

- the use of roles in a team of agents
- comparing centralised and decentralised decision making in a team of agents. This includes investigating if the DELTA decision tool (Danielson, 1997) could be used for this purpose.

The latter area is the subject of (Åhman, 1998), while the former is the subject of this thesis. It is also our intention to participate in the RoboCup 98 contest in Paris to see how our agent implementation competes with other approaches. The code is commented and some important functions are further described in this paper. I have not gone into details on trigonometric calculations etc., except in a few cases, such as the role-specific goalkeeper skills in section 8.1. At the time of writing, there is still implementation work to be done on the team. This work will be continued by the RoboCup group at the Department of Computer and Systems Sciences.

Besides describing the implementation and solutions to specific problems of the agent players, I will discuss roles from a theoretical perspective.

1.3 The Structure of this Thesis

This thesis is structured as follows: Section 2 gives some background on agent programming and classical problems in AI. The RoboCup domain is explained. Section 3 discusses roles in multi-agent systems with focus on RoboCup teams and Section 4 describes the roles used in our team.

Section 5 describes our approach to implement a RoboCup team. The usage of states and the class hierarchy are explained. A simplified object model is also presented. The most important basic and advanced skills of the players are explained in Section 6 and Section 7 respectively. Section 8 deals with some role-specific skills and behaviours of the goalkeeper. The thesis is concluded in Section 9 with a conclusion and a discussion on future work.

2 Background

2.1 Agent Programming

Intelligent agent technology originates from Distributed Artificial Intelligence (DAI), and has attracted a lot of attention and interest the last ten years. DAI is divided into research into two primary areas: Distributed Problem Solving and Multi-agent Systems (MAS). MAS research is concerned with coordinating behaviour among a collection of "intelligent agents"—how they can coordinate their knowledge, goals, skills, and plans jointly to take action or to solve problems (Bond & Gasser, 1988).

There is some confusion on the agency concept, with many different definitions of the term *agent* as a result. One of the most common definitions is Wooldridge and Jennings weak notion of agency (Wooldridge & Jennings, 1995) where an agent denotes a computer system with the following properties:

- *autonomy:* agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.
- *social ability:* agents interact with other agents (and possibly humans) via some kind of agent-based communication language.
- *reactivity*: agents perceive their environment, and respond in a timely fashion to changes that occur in it.
- *pro-activeness:* agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative.

Also noteworthy is Russell and Norvig's more general definition of agents. They state that "an agent is anything that can be viewed as *perceiving* its environment through *sensors* and *acting* upon that environment through *effectors*" (Russell & Norvig, 1995). The effectors would be arms, legs, mouth, and other body parts in the case of a human agent. Russell and Norvig split an agent into an architecture and an agent program. The agent program implements the agent mapping from perceptions to actions. This program is running on some sort of computing device, which is called the architecture.

For more extensive background information on agents, Engström and Kummeneje (1997) provide a survey over agent technology and examples of its use, and Tzikas (1997) compare agent technology and object-oriented technology.

2.2 Classical Problems in AI

Since the beginning of AI research, standard problems have been used as research domains. These problems typically deal with some restricted domain in the hope that results will scale up to other domains. Some examples of well-known classical AI problems are given below.

Blocks World. One of the most well-known and nowadays infamous AI problems is Terry Winograd's toy world domain "Blocks world". It is a simulated world made up of a number of blocks and a robot-arm capable of moving them around. Its goal is to reorder the blocks, typically building piles by stacking the blocks on to one another. The blocks world is used to exemplify the issues involved in planning systems. This

simplified domain has been criticised for not being scalable. The methodology developed in this constrained world relies too heavily on the constraints themselves. Results found in blocks world and many other toy worlds are not transferable to a less restricted situation, such as the real world.

Many of the classical AI problems are associated with game theory (von Neumann & Morgenstern, 1944). In (Miller & Cliff, 1994) game theory is described as follows:

> Game theory is concerned with analysis of formal situations called “games” where: (1) players can choose different strategies that determine their actions under particular conditions; (2) conditions and outcome unfold through the interactions of the players’ strategies; and (3) players have preferences among outcomes.

The following standard AI problems fit into this category.

Computer Chess. Chess was one of the earliest games for which programs were designed. Constructing computer systems with the ability to play chess at the level of the world's human chess champion has been a grand challenge for AI for more than 40 years. The state-space of chess is extremely large, and therefore makes it interesting for research purposes. There are now chess-playing computers that no humans can defeat. A famous example is the Deep Blue computer that won over world chess champion Kasparov in 1997 (Clark, 1997). This achievement is however not significant for other areas of AI research (Coles, 1994). The solutions are thus far of brute force nature and are merely interesting applications of parallel computing and heuristics. Just as with blocks world, computer chess does not scale up to problems in other areas, such as robotics.

Pursuit-evasion Games. In a pursuit-evasion game, a “pursuer” is chasing an “evader” in some more or less complex environment. In nature, pursuit and evasion behaviours are common because conflicts of interest over approach and avoidance are common amongst animals. This domain is thus considered ideal for investigating adaptive behaviour at many levels over many time-scales (Miller & Cliff, 1994) by constructing pursuer and evader robots or programs. Many potential robot tasks are believed to be essentially pursuit-evasion problems, e.g., obstacle avoidance and goal-directed navigation. In spite of being a restricted domain, achievements in the pursuit-evasion area seem to be transferable to some other domains because of the real-time nature and dynamics of the situation. Many results are however not applicable since significant assumptions about perfect information and the number of participants are made. This contrasts with the conditions of the domain RoboCup described in the next section.

2.3 RoboCup

The Robot World Cup, RoboCup, has been proposed (Kitano et al., 1995) as a new standard problem for AI and robotics research. A soccer game is used as a platform for a wide range of research, such as design principles of autonomous agents, multi-agent collaboration, strategy acquisition, and real-time reasoning. RoboCup is a task for a team of fast-moving robots which collaborate to solve dynamic problems. RoboCup offers a software platform, called Soccer Server, that forms the basis of the synthetic agent league for research on the software aspects of RoboCup. The idea is that agents

will have to learn to play soccer strategically. The state-space of the game is too large for anyone to hand-code all possible situations and agent behaviours.

2.3.1 Synthetic Agent League

The Soccer Server is a system that enables programs in various programming languages to play a match of soccer against each other (Noda, 1997). The system has a client-server architecture, where the Soccer Server provides a virtual field (illustrated in Figure 1) and simulates all movements of the ball and of the players. The clients are agent programs that are the brains of the soccer players. A client controls a player's movements and actions by sending commands to the server. It also regularly receives visual and auditory sensor information from the server. A client is allowed to control only one player. The communication between client and server is handled via UDP/IP sockets, allowing users to create client programs in any programming language with UDP/IP interface support.

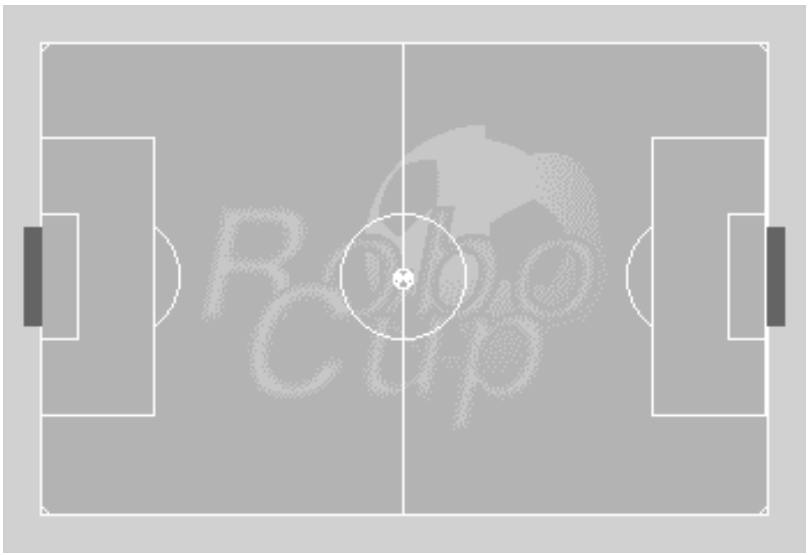

FIGURE 1. The RoboCup soccer field.

The clients are not allowed to communicate directly with each other. All communication must be done via the Soccer Server using the commands for speaking and hearing. One reason for this is to make communication between players more realistic. Players of opposing teams are provided with the same auditory information—just as in human soccer. A second reason for the restricted communication is to make evaluations of multi-agent systems on the efficiency of communication, amongst other criteria. RoboCup is thus a relatively realistic game of imperfect information.

The virtual soccer field and all objects on it are 2-dimensional. Players and the ball look like, and are treated as, circles. Landmarks, such as flags, lines and goals, are shown on the field. A client uses these landmarks to determine its player's position, as clients are not given their absolute positions but only relative positions to other objects. For more details on the regulations of the synthetic agent league and how this league differs from human soccer, see (Kitano et al., 1995).

2.3.2 Challenges

The list of challenges with RoboCup can be made long: the dynamic environment, the limited perception of each player, the possibility of having different roles for each player, team strategies, the limited communication between players, etc.

Collinot et al. (1996) state that the main difficulty in designing a software agent soccer team is "to express locally at the agent level the behaviours allowing to obtain the achievement of the collective task by the team". The team's collective task is winning the game by defending its half of the pitch and scoring.

For a more extensive list of the technical challenges targeted in RoboCup for the next 10 years, see (Kitano et al., 1997).

3 Roles

The following discussion about roles is written from a multi-agent system point of view, with a focus on the RoboCup domain.

Throughout this section the term *situation* is used. Situation will here denote a set of conditions that must be fulfilled for an agent to choose a certain behaviour. An example of a situation in RoboCup could be that a player is within the opponent penalty area and has possession of the ball. A logical behaviour in this situation would be an attempt to score.

3.1 The Usefulness of Roles

3.1.1 Reduced Complexity

First of all, it is important to understand why roles are needed in a complex multi-agent system (MAS). Consider a real-time MAS where the agents are striving towards some mutual goal. Every agent would have to analyse the situation of its environment constantly. For a role-less agent, there would be lots of different reactions and behaviours to choose from at any given time. The more options, the more difficult and time-consuming is the decision. Such an agent would be very complex. Also, there is no guarantee that the agent does not choose an action conflicting with the action of some other agent. The worst-case scenario would be that all agents in a soccer team decide to run after the ball—very messy and inefficient. The remedy here is communication between the agents. Unfortunately, communication is time-consuming and also increases complexity.

An agent with a certain role will have less behaviour options to choose from. The ideal is to have only one or two possible behaviours for every situation so that it will be easy for the agent to decide what to do. Such agents are reactive, with more or less hard-coded actions for different situations.

For the programmer, it is very difficult to foresee all the situations a role-less agent could find itself in. There is a risk that certain situations are not covered in the agent's code, which could result in erroneous behaviour. In the role-agent case, many possible

situations can be excluded. For example, a goalkeeper agent would never find himself to be in the opponent penalty-area and thus does not need behaviours for such a situation.

Without using roles there are two ways to proceed if all situations should be properly covered. The first method is straightforward. The division into situations is made coarse so that the number of situations can be kept low. The usefulness of having such general situations is however questionable as the number of behaviours to choose from in each situation would remain high. The second and preferred method is to hierarchically subdivide the situations from the first method into well-defined logical parts, perhaps using classes from object-orientation. It should thus be possible to cover all situations. This is a demanding task for the programmer. Each agent will be very complicated and the set of situations will be difficult to survey.

Roles thus reduce complexity in multi-agent systems. Reduced complexity implies that less reasoning is necessary. Agents with roles are simpler than role-less ones—you could also call them less intelligent. On their own they are not of much use, but in a MAS, a clever combination of roles can make the system very effective, fast, and robust.

3.2 Relying on Knowledge about Roles

It is an interesting question how much an agent in a team needs to know about the roles of other agents. Burkhard et al. (1997), responsible for the winning team of RoboCup'97, AT Humboldt, state the following:

> Cooperation between team partners emerges by relying on the behaviour of team-mates, that means team-mate modelling.

To what degree a team-mate needs to be modelled can be debated. An agent which knows the exact behaviour of different roles can make predictions about the behaviour of his team-mates and use this information for advanced collaboration–provided that the agent is aware of which role is associated with each agent. With such knowledge about other agents' behaviour, the need for explicit communication between team-mates can be minimized.

A RoboCup agent can take advantage of the knowledge about the roles of other agents either by using hardcoded strategies or by predicting another agent's behaviour. A hardcoded scheme could be something like "the left forward agent will always shoot the ball towards the point (X,Y) if the agent is N meters away from the goal". The right forward team-mate with knowledge of this could then have a matching scheme like "place yourself at the point (X,Y) and wait for the ball whenever the left forward gets the ball in possession". This hardcoded information is just a dumb rule, which is susceptible to unexpected events. When using prediction on the other hand, the agent models its team-mate one step further. The agent attempts to put itself in the situation of the team-mate and to determine what action it will choose to take. This implies that an agent must have the behaviour of all other roles built in.

There are many difficulties in trying to predict the behaviour of another agent. It is easy to make misjudgements about another agent's situation, since two agents will rarely have the exact same information to base their decisions on. Consider a RoboCup player

p_1 which sees an object that player p_2 cannot currently see. With the current implementation of the Soccer Server, an agent cannot tell which way another agent is looking. It would be very difficult for player p_1 to make an accurate prediction of player p_2's behaviour if p_1 is unsure of what p_2 perceives. However, it is possible to make reasonable guesses, e.g., if an agent is moving in a certain direction, it can be assumed to be looking in this direction.

In our team, agents rely on the behaviour of others for skills possessed by all players, but not for actions specific to a certain role. One reason for this is that we want to be able to design different variants of the same role and use any of them in the team without having to re-program all other players relying on one of the versions. The advantage of this careful approach is increased flexibility of the team composition. The drawback is some limitations in the sophisticated team-work.

3.3 Run-time Role Switching

The type of role-based agent in a MAS described in (Ch'ng and Padgham, 1997) can switch roles at run-time. This agent holds a whole set of possible roles and selects one based on the situation and other agents' roles. This requires that team-mates either communicate their current roles to each other or that the agent masters the complex task of inferring the role of a team-mate.

These problems are dealt with in (Stone & Veloso, 1997). They have proposed a RoboCup team structure based on formations. A formation is the arrangement into positions of players on the field, where the formation 4-3-3 places 4 defenders, 3 midfielders and 3 forwards on the field. In the proposed team structure, agents can flexibly switch roles within formations and the agents can change the team formation dynamically. It is however a difficult problem for the agent to determine if and when it should switch role. Stone and Veloso have tackled this problem by making the agents agree on procedures for coordinated role switching before the match. The formation switching of the team is based on the current score and time–variables that each agent is able to keep track of independently. The agents thus change formation simultaneously.

A simplified role-switching agent might switch role depending on the situation, without regard for other agents' roles. In such a MAS, it would be unsafe for an agent to rely on the roles of other agents. The agent would still need to know the role of its team-mates to predict their actions. The agent cannot rely on the behaviour of another agent that switches to unknown roles now and then.

3.3.1 Advantages

There are several advantages with a role-switching team over a team with rigid roles. A role-switching team is more flexible to changes, such as an agent becoming dysfunctional. Role switching saves player energy and allows players to respond more quickly to the ball (Stone & Veloso, 1997).

3.3.2 Drawbacks

As already mentioned, it is a problem to determine if and when agents should switch roles and formations. Without hardcoded switching schemes the agents in a team can become uncoordinated. For instance, an unfortunate situation where all agents in a soccer team switch to offensive roles, leaving a non-existent defense, might occur.

3.4 Default Behaviour

In a role-based soccer team, each role can have a default behaviour which is active whenever, e.g., the ball is far away from the agent. This would typically be to stay at some strategical position defined for this role and supervise the ball's movements. In a role-less team a similar default behaviour is *possible*, but there are difficulties that would need to be overcome. Strategical positions could be defined and an idle agent could look for a vacant position and place itself there. Some problems would arise though, if another agent would simultaneously head for the same position. These problems are similar to those in run-time role switching.

3.5 Teamwork

In soccer, teamwork is essential. Communication is however limited in RoboCup, so there is a trade-off between coordination by explicit communication and coordination by common tactics and strategies. Teamwork by common tactics can either be planned or emergent. Emergent teamwork is when the team members seem to cooperate "intelligently" without there being any explicit plan.

Players in the AT Humboldt team act in accordance to their expectations concerning the behaviour of their team-mates. This is a sort of emerging cooperation. The Humboldt team is an illustrative example that explicit team plans may not be necessary. These agents do not make use of explicit communication, but Burkhard et al. (1997) recognises that the teamwork would benefit from it.

In our team, teamwork is built into the actions of the players (Åhman, 1998). The agents send messages to each other to communicate intentions, e.g., asking a team-mate to pass. The positioning of team-mates is taken into consideration when an agent evaluates its current situation. The idea is that the different roles will complement each other with their different behaviours. The use of roles thus facilitates teamwork.

4 Roles in our Team

In this section the roles used in our team are explained. They correspond directly to the roles one can expect to find in a human soccer team, assuming a 4-3-3 formation.

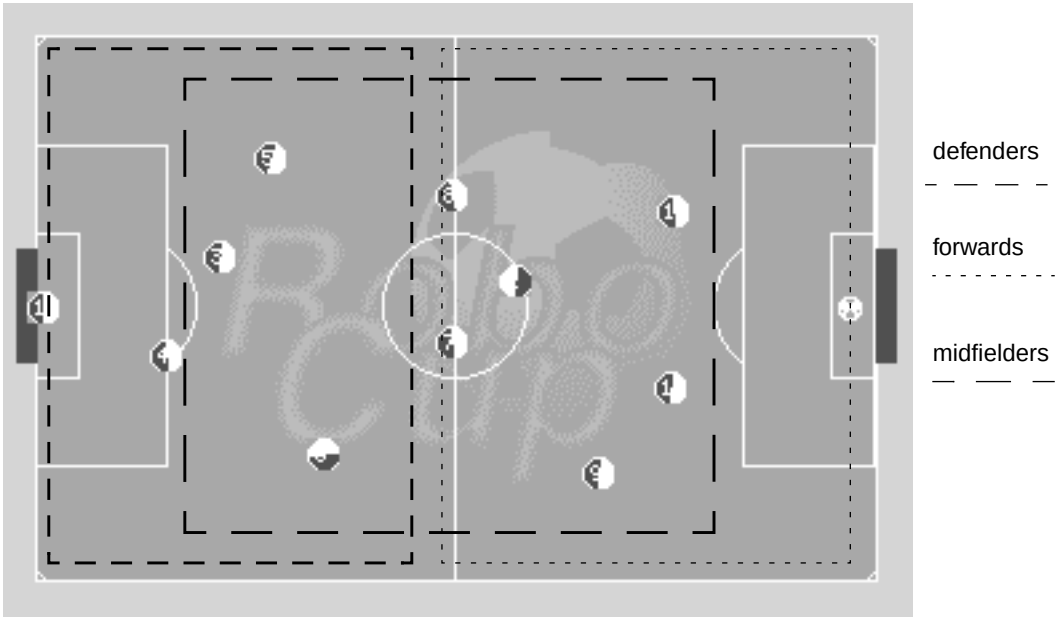

FIGURE 2. The acting areas of the different roles.

4.1 Defenders

The primary task of the defender role is to stop the opposition from scoring. A defender does this by keeping the ball out of the home penalty area and by taking the ball from opponent players. When a defender has the ball in possession it passes a team-mate further up on the field, if possible. The default position depends on which type of defender the agent is, but it will be a position inside or close to the home penalty area. The defenders always stay on their side of the pitch. The *goalkeeper*, described in section 5.4, is a special kind of defender. Another defender role is that of the *sweeper* agent which plays closest to its own goal behind the rest of the defenders.

4.2 Forwards

The agents with forward roles work together to try and score goals. They work in front of the rest of the team and stay on the opponent side and on the center of the field. The forward line consists of a *left winger*, a *right winger* and a *striker.*

Wingers are outside forwards which play to the sides of the striker and whose primary task is to provide the striker agent, or some other well-placed team-mate, with accurate passes so it can shoot at the goal. Of course, a winger can shoot directly at the goal if it has a good scoring opportunity.

A striker agent plays towards the center of the field and attempts to score once it has the ball in possession. In a more offensive team formation, such as 2-4-4, there are two strikers.

4.3 Midfielders

The midfielders link together the offensive and defensive functions of the team. Agents with midfielder roles play behind the forwards and usually stay in the center area of the pitch, although a midfielder will move over most of the field if necessary. The default position depends on if the agent is a left, center or right midfielder, but will be close to the center line. A midfielder which gets hold of the ball will dribble upwards, pass a team-mate or even attempt a goal shot if the situation is suitable.

4.4 Goalkeeper

An agent with the goalkeeper role is responsible for not letting any balls into the goal. The goalkeeper's default position is right in front of the goal and this agent stays close to the goal at all times. If it by accident leaves the penalty area, it will immediately return.

The goalkeeper has a few special skills for determining if a shot could possibly enter the goal and for intercepting goal shots. These skills are dealt with in section 8.1.

5 Implementation Issues

In this section our approach to implementing RoboCup agents is described. We use roles and an object-oriented architecture where an agent acts in accordance with its state.

5.1 State-based Representation

At every time-step of the game, an agent will be in a certain *state* based on its previous actions and its observations of the play. Each state corresponds to a behaviour where the agent tries to accomplish something: to pass the ball, avoid an obstacle, position itself, etc.

An agent has a default state which is called *play.* This state is associated with the default behaviour *optimize position*. When entering this state the player returns to a role-dependent strategic default position and observes the ball's movements.

There is no one-to-one correspondence between the specific skills, described in later sections, and a behaviour. A behaviour often consists of a sequence of skills put together to accomplish something. For instance, in the *optimize position* behaviour the skills *observe object* and *run to position* are combined.

5.1.1 State Decision

At every time-cycle of the game the player will make a state decision. This is the process of checking which state conditions are fulfilled and either remain in the same state or enter a new one. There are two ways in which an agent can terminate a behaviour and change state:

- The behaviour has accomplished its subgoal, e.g., there is no reason for an agent which has just passed the ball to a team-mate to remain in the passing state.
- Some condition of the current state is no longer valid, e.g., an agent cannot remain in the *dribble to position* state if it no longer has the ball in possession.

The default is that the agent switches to the *play* state when an action has been completed. The state decision process is not always straightforward. Rather often, there will be several states possible for the agent to enter. It is tricky problem to pick the "best" state. The aspects of decision making in such situations are discussed in (Åhman, 1998).

5.1.2 Substates

Some states involve substates which the agent will enter to perform certain actions as part of a behaviour. After these actions are completed the agent will return to the previous state. An example of a state which is usually a substate is the *obstacle avoidance* state. This substate is usually entered into when the agent is moving in accordance with the behaviour of another state and needs to avoid an obstacle in its way. The substates should not be confused with the steps that a behaviour is usually divided into.

5.2 Object-oriented Architecture

Object-orientation is used to construct the players, introducing several levels of abstraction to facilitate understanding and future extensions. More general players are superclasses of more specialised players with specific roles. Only the subclasses at the bottom of the inheritance tree may have instances, so all superclasses are abstract classes.

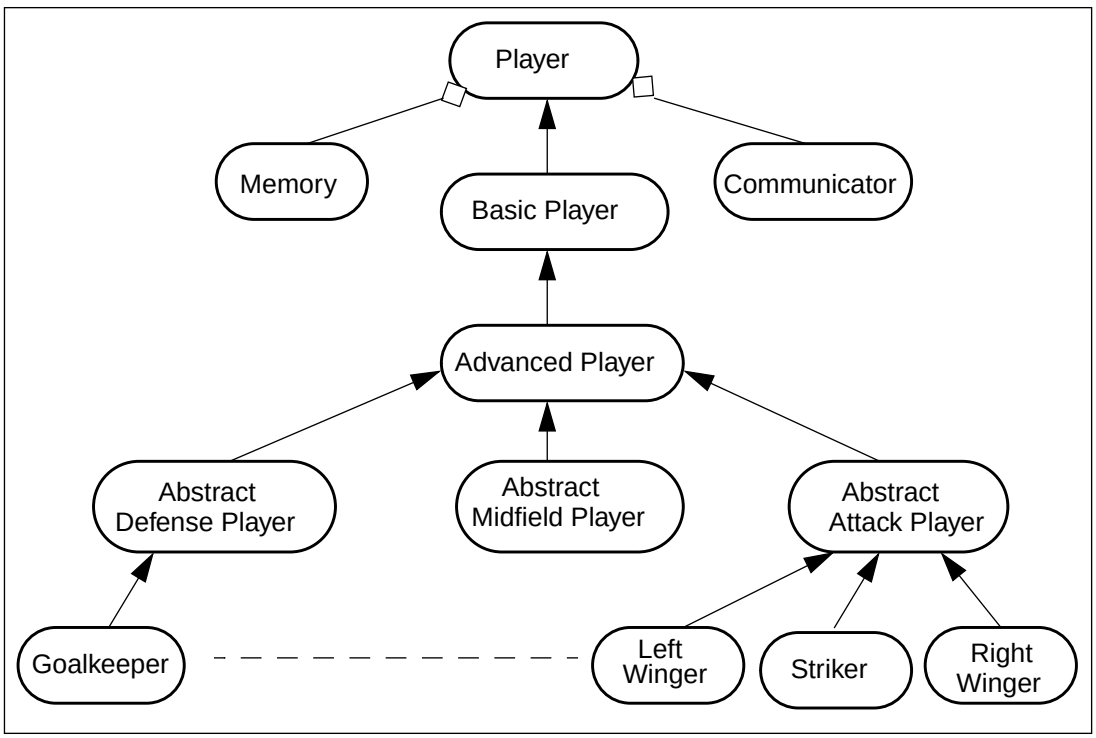

FIGURE 3. The object model. The dotted line indicates more role-specific subclasses which have been left out in this figure.

The specialised players inherit skills from the superclasses but have their own behaviour, i.e. different states with different sets of actions and, in some cases, special skills.

5.2.1 Main Classes

Player. This is the top superclass which handles communication between the agent and the soccer server. It can send basic commands like *turn*, *dash* and *kick* to the server and receives auditory and visual information which is stored in a memory object of the Memory class. The communication with the server is handled by the Communicator class which provides an interface to the Soccer Server.

Basic Player. The Basic Player class inherits from Player and contains code for basic actions and skills. These skills are described in detail in Section 6 and include observing an object, running to the ball, estimating positions, ball collection and ball interception.

Advanced Player. The Advanced Player class contains the code for the advanced skills described in Section 7. These are skills which are based on the basic skills.

The Abstract Defense Player, Abstract Midfield Player and Abstract Forward Player are superclasses with skills that the defensive, midfield and forward players respectively, have in common. Finally, the classes for each role (Goalkeeper etc.) contain role-specific code such as default positions and some highly specialised skills.

5.2.2 Implementation Language

We have chosen Java, version 1.1.4, as implementation language. An object-oriented language is obviously suitable for our OO approach. Java is a small and simple language and it is hardware independent so that we later can run our agents on other operating systems. The fact that Java code is interpreted and hence executes slower than compiled code should not be a problem. Our agent programs are not too computationally intensive and, if needed, agents of a team can execute on different computers to decrease the load on a single machine.

We have developed our programs under the Solaris operating system (Unix) since the Soccer Server, which is written in C, needs to run on this OS. Our agents have been developed for version 3 of the Soccer Server.

6 Basic Skills

In this section, the most important functions of the Basic Player superclass are described. These are basic skills which can be regarded as building blocks for more advanced skills and behaviours.

We will use the term *object* to denote something that can generate visual information from the Soccer Server. This means that an object can be a goal, a line, a flag, a player, or the ball.

6.1 Observing Objects

The skill to observe objects is useful in many situations. Observing is looking for a specific object. An agent is observing an object when directly facing it. If the object is not in the agent's field of view, the agent looks around, i.e. rotates clockwise, until the object enters the field of view. However, if it was less than 2 seconds since the agent last saw the object, it will turn directly towards the direction where the object was last seen. This is to prevent the agent from rotating unnecessarily. It is possible that the player sees the object, but does not face it. In this situation, the player will turn to face the object directly.

6.2 Estimation of Position

One of the challenges for RoboCup players is to determine positions of themselves and objects on the pitch. The Soccer Server never gives the exact position of a player. Instead, the agent is forced to estimate positions based on its visual information, just as a real soccer player would. For this purpose, some extra landmarks have been added, such as virtual flags that mark the limits of the penalty area. The positions the agent calculates can only be estimations, since there is noise in the visual information. Numbers given are only approximations.

6.2.1 Position of the Player

To be able to estimate its own absolute position the player needs to see a goal or a flag, and a line. These are all immobile objects with, for the agent, well-known coordinates. The algorithm for calculating the position given these landmarks was invented by Itsuki Noda and has been implemented in the RoboCup *libsclient.c* library. We have translated parts of this code from C to Java.

6.2.2 Position of an Object

When an agent has estimated its own position in the field, it is a simple matter to estimate the position of other objects. The agent is given the relative distance and direction to other objects by the visual information. Geometric calculations transform these values into absolute positions.

6.3 Ball Collection

For an agent to be able to direct the ball to a target position, it needs to move behind the ball and kick it towards the target. For this purpose we use a method based on the ball collection behaviour described in (Veloso et al., 1997).

Firstly, the agent needs to compute a future position of the ball. This is accomplished by looking at the ball twice at different time steps and thereby calculating its trajectory. Depending on how fast the ball is moving, a future position on the trajectory line is picked—the faster the movement the further along the trajectory line. If the ball is not moving, the future position of the ball is of course the same as the current position.

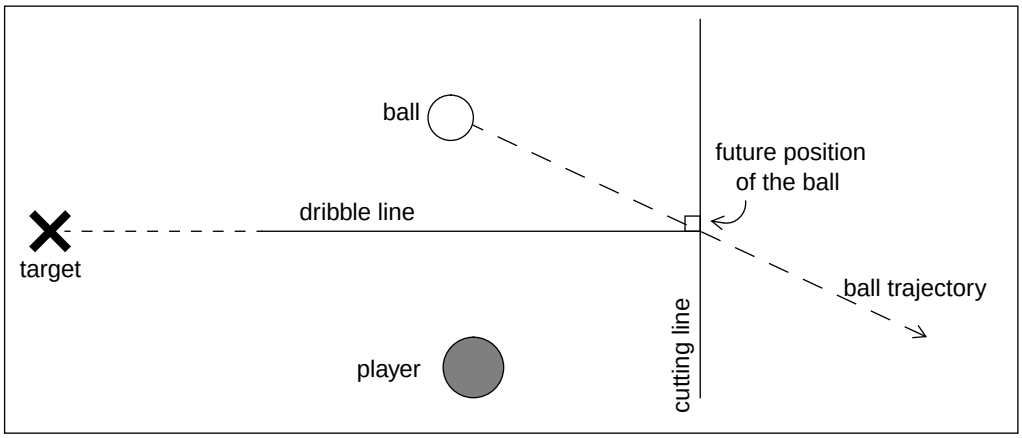

FIGURE 4. Ball collection.

Second, the agent considers a line from the target to the ball's future position, which we call the dribble line. The cutting line is the line perpendicular to the dribble line, passing through the future position of the ball. The agent needs to get behind the cutting line so that when it moves to the ball's future position it also ends up facing the target. During the time it takes for the agent to place itself, the ball will have arrived at its future position within kicking range for the agent

6.3.1 Turning with the Ball

An alternative method for the agent to move behind the ball—which has proved to be more efficient in practice—is to turn towards the target while controlling the ball. This is also the method that must be used when the target position is unknown to the agent.

The agent is assumed to have the ball in possession. At every other simulation cycle it will kick the ball very lightly sideways and then turn 20 degrees. The ball will seemingly move around the agent thus accompanying the agent as it turns around.

6.4 Ball Interception

Intercepting a moving ball is a skill used both when taking the ball from an opponent and when receiving a pass.

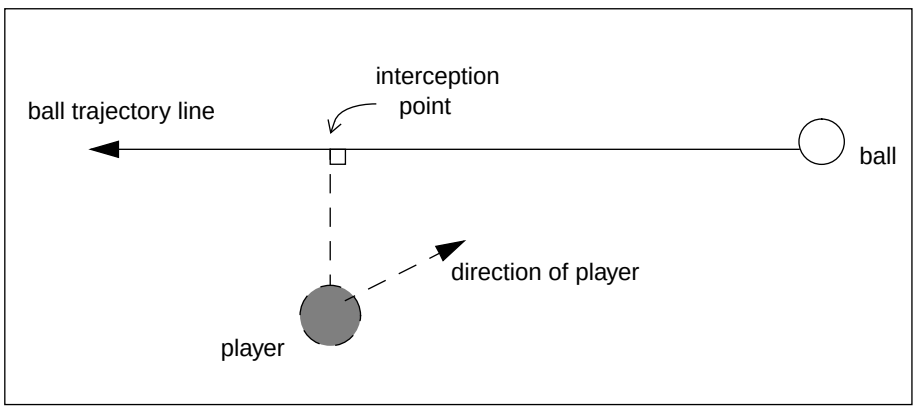

FIGURE 5. Ball interception.

To be able to intercept the ball, it must be moving in a direction towards the player. The agent then considers the line along which the ball is moving and then computes an interception point which is the perpendicular projection of the agent's position on this line. It then places itself at this point and thus stands on the ball's trajectory. Finally, the agent runs to the ball.

7 Advanced Skills

The advanced skills involve actions where players are interacting with other players and actions where changes in the environment are taken into account.

7.1 Passing

There are two different ways a pass can be initiated:

- An agent, which does not have the ball, finds itself to be in a favourable situation and requests its team-mate possessing the ball to pass.
- The agent possessing the ball finds itself to be in a dangerous situation and passes a team-mate.

Each agent thus only has to consider its own situation. These two situations are explained in detail below.

7.1.1 Pass Initiated by a Pass Request

Agent a_1, which finds itself to be closer to its present target than its ball possessing team-mate agent a_2 and observes that there is at most one opponent player between itself and the target, will request a pass. The target will in most cases be the opponent goal, but we can think of other possible targets, e.g., a team-mate that stands in a good position to score. This could make a chain of passes possible. Agent a_1 then shouts a message to agent a_2 to pass. All agents within a certain range from player a_1 will hear this message (if there is no other simultaneous shouting), but they will understand that it was not intended for them since the team number of agent a_2 is specified in the message.

If ball possessing agent a_2 hears the message it has to decide whether to pass or not. This decision is based on its present visual information. The player considers the visible *threats* in front of it. A threat could be a close opponent or a far-away opponent standing in its way. If the number of threats reaches a threshold level, the situation is considered dangerous and the player will agree to pass. If, on the other hand, the situation is considered safe or if the player is about to score, the pass request is ignored.

When the ball possessing player a_2 eventually has decided to pass, it will turn while controlling the ball (as described in section 6.3.1) towards its requesting team-mate a_1 and shoot. Then player a_2 will shout that it is passing the ball to player a_1. Agent a_1, which has been waiting for this reply, will try to intercept the ball. If agent a_2 would have decided to ignore the pass request, agent a_1 will stop waiting for a pass after a few seconds and proceed with other tasks.

7.1.2 Pass Initiated by the Ball Possessing Player

All agents continuously evaluate their situation. When the player that has the ball finds its situation dangerous, i.e. the aggregated threat has reached a certain level, it will decide to pass a team-mate. The agent then aborts its current action and starts looking for a team-mate to pass. If there are no team-mates in view, the player starts turning around while controlling the ball until a team-mate enters the player's field of view. The agent will thus pass the first team-mate it sees. However, if the agent can see several fellow team members at the same time, it will pick one arbitrarily. Of course, it would be technically possible to locate all nearby team-mates and then pass the one which is best placed. This team-mate locating would unfortunately take a while and, for obvious reasons, time is critical when the ball possessing player is threatened. It is therefore better to use a fast and simple behaviour.

7.2 Obstacle Avoidance

The skill of avoiding obstacles has been implemented to prevent players from running into things. It is difficult to dribble around an obstacle without losing control of the ball. Therefore our implementation of obstacle avoidance applies only to non-ball possessing players. A ball possessing player on direct collision course with an obstacle has other options:

- When the obstacle is a team-mate, the player simply tells the latter to move out of its way. In such a situation it is almost certain that the team-mate cannot see the other player, or else it would have avoided it. The team-mate can however hear from which direction the other player is speaking, and will use this information to move in a direction perpendicular to the sound.
- When the obstacle is an opponent, the player has entered a dangerous situation and initiates a pass, as described in section 7.1.2.

For a player not possessing the ball, obstacle avoidance is handled in the following way.

A player is always moving with a purpose, and hence has a target. This target can be either an object or a position on the field. When an obstacle within a certain distance is detected between the player and its target, the player aims to one side of the obstacle until it is in such a position that it can proceed straight to its target. The player chooses the shortest way to bypass the obstacle, which means that it will aim at the right side if the obstacle is (even slightly) to the left of the player.

Even though most collisions between players can be avoided with this behaviour, collisions do happen. In this case, the agent steps back to give itself some freedom of movement and then proceeds with the obstacle avoidance by aiming at one side.

8 Role-specific Skills

In this section some role-specific skills are introduced, i.e. skills that an agent with another role would never need. Only special skills for the goalkeeper have been implemented so far.

8.1 Special Goalkeeper Skills

8.1.1 Estimation of Possible Goal Shots

When not moving, the goalkeeper agent observes the ball. It has the special skill of computing where a potential goal shot will cross the goal line. This information could be used by the agent to place itself to block the ball or simply to decide whether it is necessary to move at all.

Calculating the goal line intersection point is somewhat complicated. The idea is simple: compute the ball's trajectory and find where the trajectory line intersects the goal line.

The goalkeeper agent can be said to have a *local* coordinate system, with its own position as the origin. The goalkeeper's view direction is the x-axis in this system. The local position (X,Y) thus denotes the position X meters in front of the player and Y meters right of the player. If the ball is in the goalkeeper's field of view, the goalkeeper receives information about the ball and its direction of movement relative to the local origin. The *absolute* coordinate system has the center mark of the soccer field as the origin. A player with the absolute position (X,Y) means that the player is X meters forward and Y meters right from the center mark, as seen from the home goal (see Figure 6). The x-coordinate is thus negative when a player stays in its own half of the field, so players of different teams have different absolute coordinate systems.

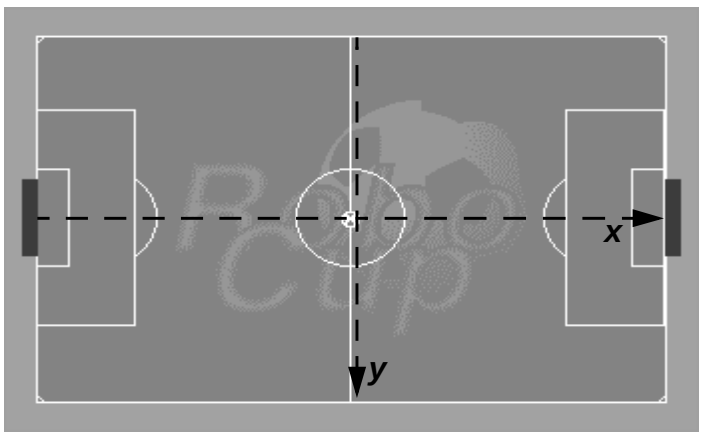

FIGURE 6. The absolute coordinate system for a player belonging to the team on the left side.

The agent knows the absolute coordinates of the goal posts and can also estimate the ball's absolute coordinates using the estimation of position skill described in the basic skills section (Section 6.2). These values are given in absolute coordinates. The goal line between the goal posts and the ball position are then transformed into the local coordinate system.

This is the transformation of a position p_A in the absolute coordinate system into the local position p_L where p_O is the absolute position of the agent, i.e. the local origin:

$p_L = M * (p_A - p_O)$ where M is the rotation matrix $M = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$

and α is the direction the player is facing in the absolute coordinate system. Similarly, the transformation from local to absolute coordinates is given by:

$p_A = (M * p_L) + p_O$ where $M = \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix}$ and $\beta = -\alpha$.

The relative movement of the ball is calculated and used to compute the trajectory line in local coordinates. The intersection point between this line and the goal line is computed and transformed back into absolute coordinates. Lines in this context means infinite lines. It is then easy to check whether this intersection point lies between the goal posts or on the extension of the goal line.

The relative movement (v_x,v_y) of the ball given visual information of the agent is calculated by a formula given in the appendix of the Soccer Server Manual (Noda, 1997).

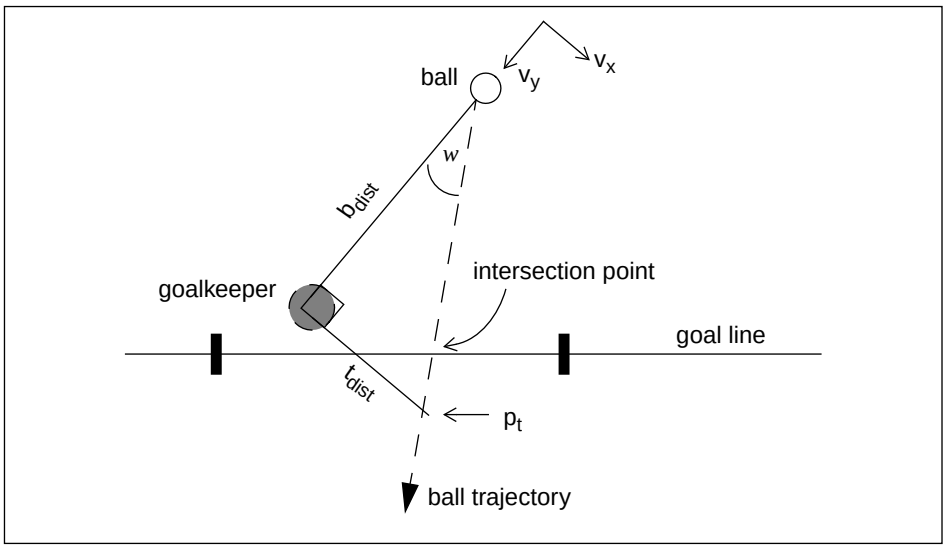

FIGURE 7. Calculating the intersection point of a goal shot on the goal line.

This is how the trajectory line of the ball is computed. The angle w in figure 5 is given by $w = \text{atan}\ (v_x/v_y)$. The agent's perceived distance to the ball is denoted b_{dist} and the distance to the future position of the ball is t_{dist}, where $t_{dist} = b_{dist} * \tan w$. The future position p_t of the ball in the local coordinate system, is therefore $p_t = (0, t_{dist})$. The latter point and the position of the ball give us the trajectory line.

8.1.2 Intercepting Goal Shots

Due to uncertainty in the agent's estimations of positions, the method above is not sufficient for determining where the goalkeeper should place itself to block the ball. The calculated intersection point is only approximate. Therefore, the agent uses another

method to actually place itself and block the ball, once it is determined that the ball might reach the goal.

The goalkeeper looks at the ball and decides if the ball is coming on its left or right side. The agent then moves a tiny bit sideways to the left or right, and then looks at the ball again etc., until it is positioned so that it is blocking the trajectory of the ball.

9 Conclusions

9.1 Tests and Results

As the decision control and the roles in our team have not been fully implemented, we have not been able to perform tests with a complete team. Tests on individual skills have however been made using “dummy players”. A dummy player is a player agent with a single behaviour hardcoded—usually the skill to be tested. For example, we have constructed simple-minded players that can only intercept balls in order to test this skill. Without these simple and frequent tests, it would have been impossible to construct any useful skills. Most of the basic and advanced skills are now thoroughly tested and are working quite well. These skills form a sound basis for implementing the individual roles.

9.2 Roles in Physical Robot Teams

Knowledge about the usage of roles in a RoboCup team can be useful to other areas since the behaviour of a software agent may be programmed into a real robot. A team of real robots could for example be sent to operate in an environment hostile to humans.

In general, it seems clear that assigning different roles to agents will be an advantage in a MAS such as a RoboCup team, and make it easier for the MAS to accomplish its goals. Run-time role switching abilities appear advantageous but are associated with some difficulties. Furthermore, role switching may only be of practical use for a MAS where agents are identical. Consider a team of physical robots. Each robot agent might have a unique physical design that is developed for its role and that helps it execute the behaviour of this role efficiently. In this case, role switching would not be a very good idea. We can conclude that role switching abilities may not always be transferable to physical agents.

9.3 Future Work

Completing the implementation work on our team with the specialised roles is one of the most important and urgent issues to get the team in a “playable” state.

Equally important is adapting the team to the latest version of the Soccer Server, version 4. Agents written for Soccer Server version 3 are not entirely compatible with the new server. The new server provides several advantages:

- A greater number of landmarks which will improve the agent’s ability to accurately determine its absolute position.

- A shorter sensing cycle, so that the agent will receive new visual information more often. Until now, the acting-sensing ratio has been 3:1, forcing the agent to make decisions based on old information.
- A *catch* command is introduced for the goalkeeper. This will make defending the goal a lot easier and make our own method for intercepting goal shots obsolete.

More improvements of the new soccer server are listed in (Åhman, 1998).

Our team is not using run-time role switching, but implementing this should be considered. Implementing agents with single roles is more straightforward and that is the reason we have chosen such an approach. The advantages with role switching are however very interesting and implementing it could be well worth the effort.

9.4 Related Work

Some related work in the RoboCup field is referred to in the Background section (Section 2) and the Roles section (Section 3). The literature on roles in multi-agent systems is surprisingly sparse. The related area teamwork is better covered, for example in (Tambe, 1996).

Most of the agent teams participating in RoboCup-97 at the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-97) did seem to use roles in one form or another. At the time of writing, all the RoboCup team descriptions and RoboCup-related papers from IJCAI-97 have not been published yet, but they will appear in *Proceedings of IJCAI-97* where the interested reader can learn more about the approaches of other soccer agent teams.

References

Åsa Åhman, *Decision Control in RoboCup Teams*, Master's thesis, Department of Computer and Systems Sciences, Royal Institute of Technology, 1998.

Alan H. Bond and Les Gasser, *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann Publishers, 1988.

Hans-Dieter Burkhard, Markus Hannebauer and Jan Wendler, *AT Humboldt - Development, Practice and Theory*, unpublished manuscript, 1997.

Simon Ch'ng and Lin Padgham, *Team description: Building teams using roles, responsibilities and strategies*, unpublished manuscript, 1997.

David Clark, *Deep thoughts on Deep Blue*, IEEE Expert, July/August issue p31, 1997.

Stephen Coles *Computer Chess: The Drosophila of AI*, AI Expert, April 1994 vol. 9 no. 4 p25(7).

Anne Collinot, Alexis Drogoul and Philippe Benhamou, *Agent Oriented Design of a Soccer Robot Team*, Proceedings of ICMAS'96, pp. 41-47, 1996.

Mats Danielson, *Computational Decision Analysis*, Ph.D. Thesis, Department of Computer and Systems Sciences, Royal Institute of Technology, Report No. 97-011, 1997.

Henrik Engström and Johan Kummeneje, *DR ABBility: Agent Technology and Process Control*, Master's thesis, Department of Computer and Systems Sciences, University of Stockholm, 1997.

Hiroaki Kitano, Yasuo Kuniyoshi, Itsuki Noda and Eiichi Osawa, *RoboCup: The Robot World Cup Initiative*, IJCAI-95 Workshop on Entertainment and AI/Alife, 1995.

Hiroaki Kitano, M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda and M. Asada, *The RoboCup Synthetic Agent Challenge 97*, to be published in Proceedings of IJCAI-97, 1997.

Geoffry F. Miller and Dave Cliff, *Co-Evolution of Pursuit and Evasion I: Biological and Game-Theoretic Foundations*, Technical Report CSRP311, School of Cognitive and Computing Sciences, University of Sussex, 1994.

Itsuki Noda, *Soccer Server Manual Rev. 2.00 (for Soccer Server Ver 3.00 and later)*, 1997.

Stuart J. Russell and Peter Norvig, *Artificial Intelligence – A Modern Approach*, Prentice Hall, 1995.

Peter Stone and Manuela Veloso, *Task Decomposition and Dynamic Role Assignment for Real-Time Strategic Teamwork*, unpublished manuscript, 1997.

Milind Tambe, *Teamwork in real-world dynamic environments*, Proceedings of ICMAS'96, pp. 361-368, 1996.

Ioannis Tzikas, *Agents and Objects: The Differences*, Master's Thesis, Department of Computer and Systems Sciences, University of Stockholm, 1997.

Manuela Veloso, Peter Stone and Kwun Han, *The CMUnited-97 Robotic Soccer Team: Perception and Multiagent Control*, unpublished manuscript, 1997.

Michael Wooldridge and Nicholas R. Jennings, *Intelligent Agents: Theory and Practice*, Knowledge Engineering Review, vol. 10 no. 2, 1995.