High level coordination of agents based on multiagent Markov decision processes with roles

Matthijs T. J. Spaan Nikos Vlassis Frans C. A. Groen

Intelligent Autonomous Systems group, Informatics Institute,
Faculty of Science, University of Amsterdam, The Netherlands
{mtjspaan,vlassis,groen}@science.uva.nl

Abstract

We present an approach for coordinating the actions of a team of real world autonomous agents on a high level. The method extends the framework of multiagent Markov decision processes with the notion of roles, a flexible and natural way to give each member of the team a clear description of its task. A role in our framework defines the set of actions and the policy of an agent. Roles are a natural way of introducing domain prior knowledge to a multiagent problem and reduce the complexity of the problem. We have applied this method in the robotic soccer domain of the RoboCup middle-size league, and present empirical results obtained at an actual tournament.

1 Introduction

In the field of cooperative robotics several people have used the concept of assigning roles to agents as a way to simplify the coordination of the team of agents. We investigate a formal way to reason about roles and present an application of them. In this paper we present an approach for coordinating a team of real world autonomous agents based on multiagent Markov decision processes. We extend this framework with the notion of roles, a flexible and natural way to give each member of the team a clear description of its task. Roles can be seen as a form of intentional cooperation as they attribute higher level capabilities to the agents and allow for heterogeneous teams. We assume that explicit communication is possible between team members to simplify coordination.

We start by reviewing multiagent Markov decision processes, extend them with roles and apply them to the RoboCup middle-size league domain. Next we present some related work in RoboCup, results from actual matches and finish with some concluding remarks.

2 Multiagent Markov decision processes

A proper way to formalize a multiagent problem is to model it as a *multiagent Markov decision process* (MMDP) [1] which is an extension of the Markov decision process framework to multiple agents. An MMDP can be described as a tuple $\langle S, N, \{A_i\}_{i\in N}, Pr, R\rangle$ where S is a set of world states, N is a finite set of agents, A_i denotes a finite set of actions available to agent i, $Pr : S \times A_1 \times \ldots \times A_n \times S \mapsto [0, 1]$ is a transition function and $R : S \mapsto \mathbb{R}$ denotes a real-valued reward function. The transition function describes the probability $Pr(s_t, \times_{i\in N} a_i, s_{t+1})$ of ending up in state s_{t+1} when executing the joint action of all agents $\times_{i\in N} a_i$ in starting state s_t. The reward function returns the reward $R(s)$ the team of agents receives in state s.

Given such an MMDP each agent should choose its action so as to maximize the expected future reward of its team. The selection of an agent's next action is determined by its policy, which is a mapping $\pi_i : S \mapsto A_i$. The policy specifies which action a_i agent i should take in state s. The joint policy π is a vector of the individual policies $\{\pi_i\}_{i\in N}$ and specifies a mapping from states to joint actions.

The task of a team of agents in an MMDP setting is to find the joint optimal policy π^* which maximizes the expected reward according to some optimality criterion. The most straightforward way of tackling this problem is to regard the group as one single agent and let some central controller compute π^*. Techniques for finding the optimal policy in single MDP settings can be applied here.

3 Roles in MMDP

Before giving a rationale why one would like to add the notion of roles to a multiagent problem we first extend the MMDP framework with the concept of roles. An *MMDP with roles* is defined as a tuple $\langle S, N, M, \{A_m\}_{m\in M}, F, Pr, R\rangle$ where S, N and R are defined as above, M is a finite set of roles, A_m denotes a finite set of actions associated with role m, $F : S \times N \mapsto M$ is a role assignment function and $Pr : S \times A_{F(s,1)} \times \ldots \times A_{F(s,n)} \times S \mapsto [0, 1]$ is a transition function. The role assignment function returns the role $F(s, i)$ assigned to agent i in state s. Note that multiple agents can be assigned the same role but no agent can have more than one role assigned to it.

Each agent i no longer has its own policy π_i but instead its policy is determined by its role m obtained using $F(s,i)$. So an agent's policy is defined as $\pi_{F(s,i)} : S \mapsto A_{F(s,i)}$, a mapping from states to actions associated with the agent's particular role. The joint policy π is a vector of the policies of the assigned roles $\{\pi_{F(s,i)}\}_{i \in N}$.

In [6] a role is defined as an abstract specification of the set of activities an individual or subteam undertakes in service of the team's overall activity. In our framework a role m records vital clues regarding the desired behavior of the agent by defining the set of possible actions A_m and by specifying the policy π_m. This way an agent can choose its actions with respect to the world state and its role without having to coordinate the actions with its teammates all the time. The roles are a form of high level coordination to prevent the necessity of having to do more coordination on a lower level. In the soccer domain roles could for instance describe a striker, a defender or a goalkeeper. The concept of assigning roles to agents is applicable in many multiagent domains, i.e. in a cleaning task where roles exist like vacuum cleaner, sweeper and mopper.

There are two main advantages of an MMDP with roles over an MMDP without roles. The first advantage is that roles are a flexible way of introducing domain prior knowledge to the problem. In soccer for instance, one might not be able to predict when the other team will attack but it is useful to have an agent in a defending role standing by just in case. In the cleaning task example one can imagine that an agent in "Vacuum cleaner" role should be restricted to vacuuming carpets while an agent in a "Mopper" role may only clean linoleum surfaces floors. Another use of roles might be in partitioning the building which needs cleaning and having each role clean a part of it.

Second advantage is that by restricting the action space to actions fit for a certain task roles reduce the complexity of the problem. In an application where rapid responses to changes in the environment are necessary this can be a crucial factor. By keeping A_m small and π_m simple one can make roles fast and more easy to build. However, a role could also be comprised of a complex policy and elaborate action set. The set of roles can be anywhere on this spectrum depending on the application domain. In robot soccer for instance the role of a striker could be simple: it needs to find the ball and try to score. A defending role however requires a more complicated policy as it should assume a good position based on the locations of its teammates and opponents.

3.1 Assigning roles

A question which remains is how the role assignment function F operates. On the one side F could impose a fixed mapping from agents to roles which reduces the MMDP with roles to one without them. On the other extreme F could pose no or little restrictions on the mapping, which results in a group of agents each pursuing the role its sees fit without any form of coordination. Both alternatives are not to our liking, we want both the flexibility of roles as well as some form of coordination in the team. F should implement a form of dynamic assignment of roles to agents, depending on the capabilities of each agent, the state of the environment and possibly the internal state of the other agents in the team.

One solution satisfying the two conditions above is to have F specify which roles in M will be assigned and in what order the distribution takes place. F determines a sequence M' of roles where $|M'| \geq |N|$. The sequence represents a preference ordering over the roles: the first role of M' is the most important role and should be assigned first, next the second role should be assigned etc. This way we are able to dynamically assign roles to agents and the team can switch configuration upon relevant changes in the world state or team makeup. The preference ordering in M' with regard to the relevance of a role to the common goal ensures the most important roles are assigned first and among the biggest group of available agents, i.e. the most important role can be assigned to any of the $|N|$ agents while for a less important role less agents are available.

Taking this approach for F requires a way to judge what agent is most suited for a certain role. For this purpose each role m has a utility function $F_m : S \times N \mapsto \mathbb{R}$ associated with it. It calculates a utility value $F_m(s,i)$ which is a real-valued estimate of the utility of agent i in role m in the current world state s, i.d. how suitable or useful would agent i be in role m given state s. F_m depends primarily on an agent's local perception of the world state but could be enhanced by information from a shared world model if available. Another component of F_m could be the agent's current role: continuing in the same role might be stimulated to counter oscillations in the switching of roles.

When a soccer robot for instance notices the ball is very close in front, its F_m will be high for those m's which describe offensive roles. For defensive roles F_m could depend on a robot's bearing and location on the field: defenders are generally close to their own goal and face the opponent's goal. If a cleaning robot is in a room with a dirty carpet it $F_{\textit{Vacuum cleaner}}$ will be high. However if a lot of dead leafs have been blown inside through an open window its $F_{\textit{Sweeper}}$ will be even higher.

For teams of heterogeneous robots F_m can differ per robot. In robotic soccer for instance the goalkeeper usually has hardware adapted for its specific task. Such a goalkeeper robot should have a very high utility for role "DefendGoal" and a very low one for the other roles. The field player robots have zero utility for role "DefendGoal" to make sure the goalkeeper robot always gets assigned the correct role. In the cleaning example different robots might have different cleaning capabilities, for instance some may have been equipped with a vacuum cleaner and a broom while others have a broom and a mop.

As a small example assume the sequence of roles M' which needs to be assigned is $\{m_1, m_4, m_4, m_2\}$ and that four agents participate in the team ($|N| = 4$). Explicit communication is used to let each robot broadcast its utility values for the roles in M', $\{F_m\}_{m \in M'}$, to the team. This way each robot can execute F to determine its own role and the roles of its teammates.

According to the preference ordering in M' role m_1 is the most important role and should be assigned first. Each agent compares its own F_{m_1} with the F_{m_1} values it received from its teammates. The agent with the highest F_{m_1} value gets assigned role m_1 and is removed from the set of available agents. Next each agent compares utility values of the second most important role in M', so in our example the F_{m_4}'s of the three remaining agents are compared and the selection is made. This process repeats until all agents have been assigned a role.

If there were only three agents ready for duty (so $|N| = 3$) m_2 would not have been assigned, but as this is the least important role it is acceptable. So F gracefully handles the case where $|M'| > |N|$ which is a necessity for some domains. In robotic soccer for instance robots are frequently removed and inserted in the game due to referee decisions or hardware failures. The ordering of M' turns out to be crucial here: if only one robot remains in the team it should assume a role as striker, with two robots a striker and a goalkeeper seem appropriate, etc.

The exact contents of M' as specified by F can be fixed for an application or depend on the state of the world as we will see in our robotic soccer application discussed below.

4 Application in RoboCup

We use a MMDP with roles to coordinate a team of robots in the RoboCup middle-size league [3]. Dynamically assigning roles to agents is a common way of tackling the problem in RoboCup (see section 6). We implemented a MMDP with roles in Clockwork Orange, the Dutch RoboSoccer team [4]. The team is a collaboration of the Delft University of Technology, the Utrecht University and the University of Amsterdam.

RoboCup middle-size league is a multiagent system consisting of two teams of four autonomous real world agents playing soccer against each other. The teams are competitive of course, but the members of each team should cooperate to achieve the common goal: score more goals than your opponents. The domain is partially observable, one agent cannot perceive the complete world state with absolute certainty, but at the moment we do not explicitly deal with this partial observability. Explicit communication is allowed but might fail and can be considered as having a certain cost. Maintaining a shared world model is in principle possible, but at a cost as it requires large amounts of communication. As the game is very dynamic and a team of opponents tries to obstruct an agent's plans, the latter should act quickly and respond timely to significant changes in the world state.

4.1 Team strategy and roles

Which roles need to be assigned is defined by the global team strategy: the team autonomously switches one team strategy to another based on ball possession. The team strategy simply determines the contents of the sequence of roles M'. Either your team has the ball and attacks, the other team has the ball and your team defends, or nobody has ball possession and your team tries to obtain it. Table 1 shows M' for each team strategy. Note that only three roles are associated with each team strategy as the goalkeeper does not actively participate in team play: it has its fixed role of goalkeeper due to its fundamentally different hardware.

A robot in role PassiveDefend (PD) waits in front of its own goal for the opponent team to attack. Role InterceptBall (IB) chases the ball trying to obtain control over it. Role ActiveDefend (AD) is a defensive variant of the previous role. When a robot controls the ball it assumes role AttackWithBall (AWB), which usually means dribbling with it toward the enemy goal followed by a shot. Role AttackWithoutBall (AWoB) describes an auxiliary attacker moving toward the enemy goal together with the main attacker.

The F_m of each role is based on two measures: first one is the time a robot expects it needs to reach the ball and the second one is how well the position of a robot is suited for the role. The first measure is relevant for the ball oriented roles AttackWithBall, InterceptBall and ActiveDefend while the second one is important for the non ball oriented roles PassiveDefend and AttackWithoutBall.

The set of actions A_m associated with each role is a subset of the infinite set offered by the lower level software responsible for executing actions. This set is infinite as actions are defined as having a type like "move", "dribble" or "shoot" and certain continuous parameters such as target position or heading. A_m should not be too big as we will take into consideration each $a_m \in A_m$ when choosing the next action, and the process should not take too much time since the robot has to respond quickly. Next to a set of basic movement, "seek ball" and "chase ball" actions incorporated in all five A_m's each role has its specific actions. For instance, A_{AWB} contains several dribble and shoot actions while A_{PD} contains move actions.

Table 1: *Distribution of roles associated with each team strategy.*

Team strategy	M'
Attack	[*AttackWithBall*, *PassiveDefend*, *AttackWithoutBall*]
Defend	[*ActiveDefend*, *PassiveDefend*, *ActiveDefend*]
Intercept	[*InterceptBall*, *PassiveDefend*, *InterceptBall*]

4.2 Real-time planning

We have not yet learned the policies π_m for all roles but started with hardwired ones. We employ a form of real-time planning: for each action $a_m \in A_m$ in state s_t find the most probable s_{t+1}, multiply the reward $R(s_{t+1})$ with the estimated probability of success to get the utility of executing a_m. Execute the action with the maximum resulting utility. If the shared world model is operational first run this scenario for each of your teammates and then for yourself to take their actions into consideration.

The world state is composed of the position, heading and speed of all known moving objects (plus an estimate of the uncertainty in these measurements), ball possession (our team, their team, nobody/unknown) and the current distribution of roles over the agents. In a continuous real world domain like RoboCup middle-size league with complex agents estimating a complete and accurate transition function Pr is intractable. Therefore we have chosen to simplify Pr: given current state s_t and action a Pr calculates only one resulting state s_{t+1} but Pr does estimate the probability of success of a. Our reward function R takes into account several soccer heuristics: whether the ball is in one of the goals, ball possession, strategic positioning and how well the location of the robot is suited for its role. The latter can be considered as part of the policy π_m.

5 Results

Clockwork Orange successfully participated in the RoboCup 2001 tournament in Seattle, USA reaching the quarterfinals and at the German Open 2002 tournament the team became fourth. At RoboCup 2001 seven games were played, resulting in three victories, one draw and three losses. At the German Open 2002 tournament Clockwork Orange played eight matches: four victories, two draws and two losses. Results regarding team coordination will be presented from the last four and a half[1] games from RoboCup 2001 and all German Open 2002 matches. The outcomes of these matches can be found in table 2.

Next we will discuss a 30 second fragment from the RoboCup 2001 match against GMD, shown in figure 1. This fragment gives an example of how our coordination mechanism works under good conditions: the world model is consistent at an acceptable degree. The robots in our team are called Caspar, Ronald and Nomada.

[1]During the half-time interval of the third game the team coordination implementation was frozen for the rest of the tournament.

At the start of the fragment Caspar has control over the ball and each robot believes the team strategy is *attack*, as they should. The team has correctly assigned the AttackWithBall role to Caspar and the AttackWithoutBall role to Ronald while Nomada is standing close to the own goal area in role PassiveDefend. Then Caspar loses the ball, the team switches to team strategy *intercept* and the two attacking robots switch to role InterceptBall. Between $t = 115$ and $t = 118$ there is some confusion whether or not the opponent controls the ball which leads to oscillations between team strategies *intercept* and *defend.*

At about the same time as Ronald gains possession of the ball Nomada is neatly shut down and removed from the game. This means the team has lost its defender and one of the other two should fill the gap. Ronald does so by switching to role PassiveDefend at $t = 124$, after having been in error for a short while: it was at the same time as Caspar in role AttackWithBall although it was really Caspar which controlled the ball.

Figure 1 shows the good performance of our team coordination mechanism under good conditions, but even then inconsistencies sometimes occur. For instance at $t = 131$ Caspar briefly switches to role InterceptBall when it shouldn't. The reason is that Caspar's world model has not received a position update from Ronald's world model for a while, as Ronald has no estimate of its own position that is accurate enough to communicate. From Caspar's point of view it is the only active member of the team and it should thus fulfill the most important role in team strategy *intercept*: InterceptBall. These kind of problems are common and should be properly dealt with.

The next table illustrates the mean times between role changes at the two tournaments together with their standard deviation and the total period of time concerned:

	mean	std dev	total (hrs:min)
RoboCup 2001	2.50	4.41	3:24
German Open 2002	2.98	4.33	6:33

A plot of the logarithm of these times resembles the normal distribution $N(0, 1)$.

When preparing for the German Open 2002 tournament we adjusted the team play of Clockwork Orange to let the

Table 2: *Results of Clockwork Orange matches on two international tournaments. Scores should read as Clockwork Orange vs. opponent team.*

RoboCup 2001		German Open 2002			
Trackies	0-8	Philips CFT	1-0	Ulm Sparrows	6-0
Fun2maS	5-0	FU-Fighters	4-0	Attempto Tübingen	3-1
Artisti Veneti	3-0	CoPS Stuttgart	0-0	CS Freiburg 2002 (semifinal)	0-7
GMD	1-4	CS Freiburg 2002	0-0	GMD-Musashi	0-3
CS Freiburg	0-4	(round robin match)			

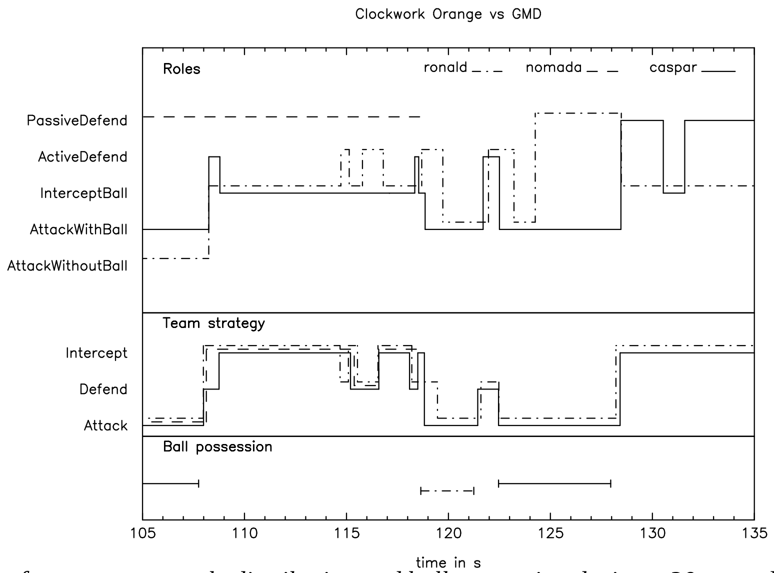

Figure 1: *Visualization of team strategy, role distribution and ball possession during a 30 second period of the RoboCup 2001 game against GMD. The x-axis displays the wall clock time in seconds since the start of the game. The robots involved are Ronald, Caspar and Nomada (removed from the game 118.88 seconds after game start). Their role and team strategy are shown on the y-axis. When one of them has ball possession a line is drawn in the bottom part of the graph.*

Table 4: *The portion of time each robot spent in each role against specific opponents.*

RoboCup 2001	PD	AD	IB	AWB	AWoB	Total (s)
Trackies	●	•	⬤	•	·	1050.45
Fun2maS	●	•	●	●	·	2256.67
Artisti Veneti	●	•	⬤	•	·	2638.13
GMD	●	•	⬤	●	·	2467.76
CS Freiburg	●	•	⬤	•	·	2843.96
German Open 2002	**PD**	**AD**	**IB**	**AWB**	**AWoB**	**Total (s)**
Philips CFT	⬤	⬤	•	•	·	2967.55
FU Fighters	●	⬤	•	•	·	2207.88
CoPS Stuttgart	●	⬤	•	•	·	3729.60
CS Freiburg 2002 (1st)	●	⬤	•	●	·	3300.57
Ulm Sparrows	●	●	•	•	·	2676.32
Attempto Tübingen	●	●	•	•	·	2286.03
CS Freiburg 2002 (2nd)	⬤	●	•	•	·	2460.83
GMD-Musashi	⬤	●	•	•	·	3314.93

Table 3: *Portion of time spent in a certain team strategy.*

	Attack	Defend	Intercept
RoboCup 2001	●	•	⬤
German Open 2002	•	⬤	•

team follow a more defensive strategy. Table 1 was not modified, but a restriction was added to the transition of one team strategy to another. Moving from Defend to Intercept was forbidden, which means the team can only leave the defensive team strategy by obtaining control over the ball. Table 3 shows the effect of this change: at the German Open 2002 tournament the paramount team strategy was Defend in contrast to the RoboCup 2001 tournament where team strategy Intercept was used most of the time.

The distribution of roles over robots can be found in table 4. The difference between the RoboCup 2001 and German Open 2002 matches is immediately clear: a shift from role InterceptBall (most important role of Intercept) to role ActiveDefend (most important role of Defend) has taken place as was to be expected after the team play adjustment. The table also shows that the AttackWithoutBall role has been assigned only a small amount of time. It is the least important role of team strategy Attack which means it will only be used if there are three field players active and one of them has ball possession.

To see the influence of a robot's role on its action selection we have included tables 5 and 6. They show what type of actions have been chosen while the robot was in a certain role. The tables demonstrate that a robot's role influences the kind of actions it takes. Chase actions, whose purpose it is to obtain the ball, are not used as much in roles PassiveDefend and AttackWithoutBall as in the other roles, which one would expect given their nature. The AttackWithBall role is the only role which uses Dribble actions often which fits with the definition of the role: try to dribble with the ball to the opponent team's goal and shoot. Even AttackWithBall only shoots a very small amount of time, but that seems consistent with human soccer.

6 Related work

Distributing roles among the field players is quite common in RoboCup middle-size league. The players of CS Freiburg distribute roles amongst themselves, namely an active, support and strategic role [7]. The active robot tries to get the ball, a support player attempts to assist by positioning itself appropriately and a strategic player occupies a defending position. Each robot determines its utility to pursue a certain role and informs its teammates. Based on these utilities a robot chooses his role.

Roles are also distributed among the players of ART [2]. The roles they define are a main attacker which demands ball possession, a supporting attacker and a defender. The protocol for distribution of the roles among the players is based on two utility functions. Every cycle each robot computes both utility functions based on its perception of the world and sends the results to its teammates. The robot with the lowest result for the first utility function becomes the main attacker while the remaining two field players compare the value of the second utility function. The one with the lowest value takes the role of supporting attacker and the other one becomes defender.

Dynamic role assignment based on utility functions seems a flexible way for achieving cooperation, but most of the standard role distributing schemes as designed by other middle-size league teams seem to specify just three roles and focus on attacking. Adding the concept of a global team strategy allows us to guide the team better in circumstances which e.g. ask for defensive instead offensive play. Our team strategy extends the concept of formation as defined in [5] by assigning priorities to the roles to cope with a variable number of participating agents.

7 Conclusions

We have presented an approach for coordinating the actions of a team of real world agents on a high level, its application on the robotic soccer domain and empirical results obtained at an actual tournament. The extension of multiagent Markov decision processes with the concept of roles seems justified: they provide a flexible solution to the problem of distributing the global task of a team among its members. Roles reduce the amount of low level coordination necessary and they clearly influence the actions an agent takes. Future work includes dealing with the partial observability of the domain and moving from hand coded policies to ones obtained by reinforcement learning.

Acknowledgments

This research is supported by PROGRESS, the embedded systems research program of the Dutch organisation for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW.

References

[1] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Theoretical Aspects of Rationality and Knowledge*, pages 195–201, 1996.

[2] C. Castelpietra, L. Iocchi, M. Piaggio, A. Scalzo, and A. Sgorbissa. Communication and coordination among heterogeneous mid-size players: ART99. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*. Springer-Verlag, 2001.

Table 5: *The portion of time spent executing an action of a certain type while in a certain role. Data is from all field player robots during the last four and a half games of RoboCup 2001.*

	Turn	Move	Dribble	Shoot	Seek	Chase	Total (s)
PD	●	●	·	·	●	●	3529.41
AD	•	●	•	·	•	⬤	1325.78
IB	•	●	·	·	●	●	4238.73
AWB	•	•	●	•	•	●	1812.27
AWoB	●	⬤	·		·	•	326.89

Table 6: *Same type of table as 5, only the data has been obtained at the German Open 2002 tournament.*

	Turn	Move	Dribble	Shoot	Seek	Chase	Total (s)
PD	●	●	·		●	⬤	5955.67
AD	•	•	·	·	●	⬤	9603.47
IB	•	•	•		●	⬤	2629.04
AWB	•	•	⬤	•	•	●	2472.25
AWoB	●	●	·		•	●	337.72

[3] M. T. J. Spaan and F. C. A. Groen. Team coordination among robotic soccer players. In G. Kaminka, P. U. Lima, and R. Rojas, editors, *RoboCup 2002*. Springer-Verlag, to appear.

[4] M. T. J. Spaan, M. Wiering, R. Bartelds, R. Donkervoort, P. Jonker, and F. Groen. Clockwork Orange: The Dutch RoboSoccer Team. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001*. Springer-Verlag, to appear.

[5] P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110, 1999.

[6] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[7] Th. Weigel, W. Auerbach, M. Dietl, B. Dümler, J. Gutmann, K. Marko, K. Müller, B. Nebel, B. Szerbakowski, and M. Thiel. CS Freiburg: Doing the right thing in a group. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*. Springer-Verlag, 2001.