Integration of Advice in an Action-Selection Architecture

Paul Carpenter, Patrick Riley, Manuela Veloso, and Gal Kaminka

Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3891, USA
{carpep, pfr, mmv, galk}@cs.cmu.edu

Abstract. The introduction of a coach competition in the RoboCup-2001 simulation league raised many questions concerning the development of a "coachable" team. This paper addresses the issues of dealing with conflicting advice and knowing when to listen to advice. An action-selection architecture is proposed to support the integration of advice into an agent's set of beliefs. The results from the coach competition are discussed and provide a basis for experiments. Results are provided to support the claim that the architecture is well-suited for such a task.

1 Introduction

In the future, the complexity of the tasks agents will be expected to perform will dramatically increase. It is not feasible to think one could hand-code execution plans for agents to utilize in all situations. Instead, it is likely that agents will employ a generalized architecture capable of integrating advice from external agents tailored to its current goal. Advice is not necessarily a plan to reach a goal, but rather hints or directions that will likely aid in the agent's planning.

In the simulated robotic soccer domain, an online coach acts as an advice-giving agent [11] [15]. The coach receives a global view of the world but it has limited opportunities to communicate with the team. Therefore, it is impossible for the coach to act as a centralized agent controlling the actions of the other agents. However, a coach has the ability to periodically offer general advice or suggest changes in the team's strategy. It is clear that the coach has the potential to greatly impact the performance of a team; whether it is for better or for worse depends on how the team chooses to use the advice.

This work describes the issues one must consider while implementing a "coachable" team. In addition, this work describes how coach advice is integrated into the ChaMeleons-2001 action-selection architecture as well as how advice from other agents (e.g., a "team captain", others teammates, or even humans) can be used [2].

A specific issue of consideration is whether to blindly follow all advice that is given. For example, the ChaMeleon's online coach, OWL, offers advice in the form of passing rules [14] [13]. Consider the situation when an agent has a clear shot on goal but one of the coach's passing rules is applicable - should the agent follow the advice and pass the ball or should it attempt to score on the open goal? In the same situation, a human is able to quickly reason about the choices and realize that an open shot on goal must be an exception to one of the coach's rules. How does an agent know when to ignore the coach however? The coach agent could attempt to make its advice rules more strict, incorporating exceptions in the rules; however, this is not feasible. Instead, the players need a mechanism to recognize exceptions to the advice given to them.

An agent should also be allowed to ignore pieces of advice in order to resolve conflicting rules. As in the example above, a coach gives a rule for player one to pass to player two given a specific state of the world. Later, the coach observes that player one should shoot on goal in that very same state. Player one now has two conflicting pieces of advice and cannot execute both actions simultaneously. If it chooses to follow the advice of the coach, it must resolve this conflict and choose only one rule to follow.

A third issue that must be considered is how to handle advice in such a way that facilitates the inclusion of advice from several sources. The advice could be weighted differently or organized as a hierarchical mixture of experts similar to [6].

The design of the ChaMeleons-2001 addresses each one of these issues.

2 Related Work

There has been little research involving online coaches in the RoboCup simulation community. Before RoboCup-2001, the most common uses of a coach were to communicate formation/role information [1] [3] [4] and setplay information [4] [12] [16]. Robocup-2001 introduced a standard coach language in which information is communicated to the players via condition-action rules [15]. With the introduction of the coach competition in 2001, it is likely that more will concentrate on developing "coachable" teams.

The idea of integrating advice from an external source is not new though; McCarthy made the suggestion nearly 45 years ago [9]. However, the number of systems that have done just this are limited and usually specific to a particular problem, e.g., integrating advice in Q-learning to increase an agent's overall reward [7].

There has also been work that attempts to address how to handle conflicting advice [5]. This is one of the very same issues soccer agents must address when accepting advice from other agents. [5] offers four criteria to determine how to resolve conflicts:

- Specificity - more constraining rules followed first
- Freshness - more recent advice followed first
- Authority - one agent has the ability to over rule another agent's advice
- Reliability - follow the advice that has the highest probability of success

This technique is much different than how conflicts are resolved in ChaMeleons-2001. Depending on the type of conflict, advice may be ignored or prioritized based on what type of advice is given. Modifications to the architecture to support other types of complex conflict resolution, e.g., the ones proposed by [5], would require implementing only a new behavior arbitrator.

One of the techniques used by the ChaMeleons-2001 to solve conflicts is very similar to the soft enforcement method introduced by [10]. Soft enforcement gives preference towards plans consistent with pieces of advice. Advice that introduces conflicts however are ignored. In the ChaMeleons-2001, before advice is attached as a child, it is verified as being both recommended and not discouraged by the coach.

3 Architecture Overview

The ChaMeleons-2001 utilize a hierarchical, behavior-based architecture. Action-selection for the agents presents itself in the three primary components of the architecture:

- Individual Behaviors
- Behavior Manager
- Behavior Arbitrator

An agent executes the root behavior each cycle. The root behavior selects which actions to consider depending on the current world state. The Behavior Manager is responsible for actually instantiating the behaviors to consider as well as including other considerations based on the coach's advice. A Behavior Arbitrator is used to ultimately determine which choice is executed. The relationship between each component is shown in Figure 1.

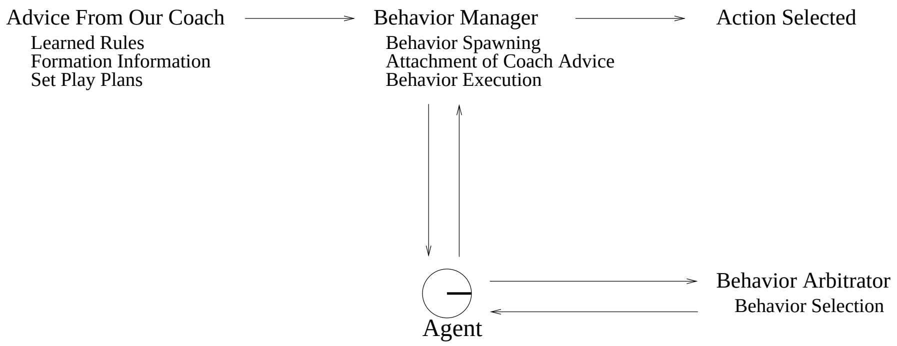

Fig. 1. Action Selection Architecture

3.1 Behaviors

A behavior is a means to achieve a goal. The goal is reached through the execution of a behavior and a chain of child behaviors. A child behavior is a way to achieve the goal of its parent and it too may have children of its own. Every child of a behavior is an alternative way to achieve the goal of its parent. For example, the *passBall* behavior may have many *passToPlayer* children corresponding to the different passing options it has. Every behavior also possesses the following properties:

- **Applicability Conditions** - Determines whether or not it is appropriate for the behavior to execute given the current state of the world
- **Probabability of Success** - The likelihood that the behavior will execute successfully
- **Value** - The value of the future world state assuming its execution is successful
- **Class** - Used when integrating advice by adding those behaviors recommended by the coach that match one of the classes of its children. Therefore, a *passToPlayer* behavior recommended by the coach is added as a child of *passBall* because its class matches one of *passBall*'s children.
- **Source** - A behavior is generated by its parent or as a result from advice. A behavior's source marks its origin, i.e., parent or coach

Behavior Organization The behaviors are organized in a hierarchical fashion with the most primitive actions being at the bottom and the root behavior at the top. The hierarchy is organized as a directed acyclic graph with vertices and edges (B, E) where:

$$B = \{b : b \text{ is a behavior}\}$$
$$E = \{(b_1, b_2) : b_2 \text{ is a child of } b_1\}$$

A behavior's set of children can be defined in terms of the DAG as:

$$\text{Children}(b) = \{c : (b, c) \in E\}$$

A primitive action is a behavior at the lowest level of the hierarchy as it has no children. Each primitive action in the robotic soccer simulator corresponds to one of the possible actions to send to the server, e.g., kick, turn, dash [15]. The set of primitive actions is defined as:

$$A = \{a : a \in B \wedge (\neg\exists b)((a, b) \in E)\}$$

Given a set of primitive actions, the DAG has the property that there exists a path from every behavior to at least one primitive action:

$$(\forall b \in B)(\exists a \in A)(\exists p)(p \text{ is a path from } b \text{ to } a)$$

Because there exists a path from every behavior to one of the primitive actions, it is guaranteed that at every simulation cycle, an action will be selected to execute.

3.2 Behavior Manager

The Behavior Manager (BM) is the component responsible for all things related to a behavior's creation and execution, including the integration of advice. The BM maintains a pool of fresh behaviors so that they do not have to be repeatedly instantiated throughout the action-selection process or even across cycles. Stale behaviors are periodically flushed from the pool in such a way to balance the time needed to search for behaviors in the pool and the time required to instantiate new behaviors.

Behaviors are also executed through the BM. The BM logs the execution state of each behavior and if no action is selected, it may attempt to execute another behavior [17]. The bookkeeping the BM maintains makes it possible to trace the complete chain of behaviors considered and executed. As a result, the architecture used to implement a "coachable" team is also convenient in tracing and debugging behaviors during development.

Integration of Advice The BM is also responsible for integrating advice from the coach. Advice generates new behaviors to be added to a behavior's set of children. Given a set B_{advice} of coach recommended behaviors, the advice integration is done as follows:

$$\forall b \in B \exists b_c \in \text{Children}(b) \exists b_a \in B_{\text{advice}} b_c.\text{class} = b_a.\text{class} \Rightarrow E \leftarrow E \cup \{(b, b_a)\}$$

A behavior recommended by the coach is inserted into the set of children of all behaviors for which it matches one of the behaviors' classes of children. The integration of advice from other sources would be done in the same fashion. Because all behaviors maintain its source as one of its properties, different arbitration methods, e.g. a hierarchical mixture of experts, could potentially be developed to decide among the children [6].

3.3 Behavior Arbitrator

A Behavior Arbitrator (Arb) is a function that chooses a single behavior to execute among a behavior's set of children. More specifically, given a set of children a behavior is considering for execution, an arbitrator determines which child to actually execute. In general, an arbitrator selects the argument c that maximizes some function $f(c)$:

$$\text{Arb}(b) = \text{argmax}_c \; c \in \text{Children}(b) \wedge f(c)$$

The type of arbitrator used depends highly on the class of the behavior, however they all maximize some function that "scores" each child. Three arbitrators are described below:

1. **Value-Based** - The child that maximizes the future state of the world:

$$\text{Arb}_{\text{value}}(b) = \text{argmax}_c \; c \in \text{Children}(b) \wedge c.\text{value}$$

2. **Probability of Success** - The child with the highest probability of success:

$$\text{Arb}_{\text{prob}}(b) = \text{argmax}_c \; c \in \text{Children}(b) \wedge c.\text{probabilityOfSuccess}$$

3. **Coach-Based** - If there exists children recommended by the coach, arbitrate among those choices. Otherwise, choose an alternative arbitration method:

$$\text{Arb}_{\text{coach}}(b) = \left\{ \begin{array}{ll} \text{argmax}_c \begin{array}{l} c \in \text{Children}(b) \wedge \\ c.\text{src} = COACH \wedge \\ f(c) \end{array} & , \{c : c.\text{src} = COACH\} \neq \emptyset \\ \text{Arb}(b) & , \text{otherwise} \end{array} \right\}$$

The example arbitrators are all fairly straightforward. The complexity of the arbitration depends highly on the scoring function, $f(c)$. One could potentially use decision trees to arbitrate among passing options [17] or neural networks to determine when and where to shoot on goal [8]. One of the more complex arbitrators used by the ChaMeleons-2001 is the *handleBall* BA.

handleBall Arbitrator The *handleBall* arbitrator provides a way for the players to choose what behavior to execute while having possession of the ball, including when, and when not, to follow the advice of the coach. There exists a set, P, of priority levels each having a behavior descriptor. Associated with each priority level is a threshold for the behavior's probability of success. The probability threshold, currently hand-coded, must be met in order for the behavior to execute. When given a set of choices, the behavior with the highest priority that meets its threshold is executed. A subset of the priority levels and thresholds is shown in Table 1. Please note that it is possible for the same descriptor to appear with different priorities and thresholds. This allows a behavior to be considered for execution at more than one priority level based on its threshold of success. For example, if a shot on goal has a .6 probability of success, it has a fairly low priority, however, it still might have a higher priority than simply clearing the ball down the field. A shot with a .8 probability of success has a much higher priority and therefore would be one of the first behaviors considered for execution.

There exists a relation, R, from *handleBall*'s set of children, C, to the set of priority levels, P. It is possible for a behavior to be described by more than one priority level. For example, *pass_forward* and *pass_to_less_congested* could potentially describe the same behavior. The relation has the property that the set of behaviors a priority level describes all have the same type:

$$\begin{array}{l} \forall p \in P \exists b_1 \in B \exists b_2 \in B \\ (b_1, p) \in R \wedge (b_2, p) \in R \wedge b_1 \neq b_2 \Rightarrow b_1.\text{class} = b_2.\text{class} \end{array}$$

In order to choose a single child to execute, the elements in the set of priority levels are sorted based on the priority of each descriptor:

$$(\forall p_i \in P)(p_i < p_{i+1})$$

Priority	Descriptor	Success Probability Threshold
1	*shoot_on_goal*	.8
2	*coach_pass_for_shot*	.7
3	*pass_for_shot*	.8
4	*pass_forward*	.75
5	*dribble_to_goal*	.75
6	*dribble_to_corner*	.8
7	*pass_to_less_congested*	.7
8	*coach_pass_forward*	.8
9	*pass_to_closer_to_goal*	.75
10	*pass_to_better_path_to_goal*	.8
11	*shoot_on_goal*	.6

Table 1. One Possible Priority Level Ordering

This allows the arbitrator to be defined as a function of a behavior's probability of success (probSucc) and its threshold:

$$\text{Arb}_{handleBall} = \text{argmax}_{i,b} \begin{array}{l} b \in \{b : (b, p_i) \in R\} \wedge \\ p_i \wedge b.\text{probSucc} \wedge \\ b.\text{probSucc} > p_i.\text{Threshold} \end{array}$$

The *handleBall* arbitrator maximizes both the priority level and the probability of success for all behaviors at that level, given the constraint that the threshold is met. The ordering of the priority levels makes it possible to ignore the coach when it is possible to execute a behavior that is given a higher priority. For example, an agent should always choose to shoot when the probability of success is high as opposed to making a coach-recommended pass.

4 Coach Advice

This section gives an overview of the type of advice given by the OWL coach, which was also created at Carnegie Mellon [14] [13].

There are five main types of advice. The first four are given at the beginning the game. The first gives a home position for each member of the team to define a team formation. Next, static marking advice assigns defenders to mark the opponent forwards, based on the expected opponent formation. Passing advice is based on trying to imitate another team. Rules are learned about passing behavior of a team by observing them. Those rules are then sent as advice to try and mimic the observed team's behavior. Similarly, rules are learned about the opponent's passing, and the advice the agents receive predicts the opponent's passes. Finally, during the game play, the coach makes short passing and movement plans for "set-play" situations, where our team has a free kick. For information concerning how the advice is generated, please refer to [14] [13].

5 Experimental Results

The RoboCup simulation league had its first coach competition in 2001. The competition was open only to teams that supported the standard coach language [15]. Teams played a baseline game without a coach against a fixed opponent. A team's coach is used with all the other teams against the same opponent, Gemini, and the winner is determined based on a ranking of goal differentials. The teams were not tested using their own coaches since the competition was geared towards developing a coach capable of improving the performance of any team. The goal differentials were determined by differences in goals scored, e.g., the goal differential for the Helli-Respina game using the Wright Eagle coach is -14 (3 fewer goals scored, and 11 more goals allowed).

	Coach Used				
	No Coach	Wright Eagle	Helli Respina	Dirty Dozen	ChaMeleons-2001
Wright Eagle	8:0	-	0:0	0:0	2:0
Helli-Respina	5:0	2:11	-	0:14	0:0
Dirty Dozen	0:10	0:28	0:29	-	0:20
ChaMeleons-01	0:4	0:6	0:6	0:9	-

Table 2. Results of the 2001 RoboCup Coach Competition

The results of the coach competition, given in Table 2, were very surprising. It appeared as if using a coach hindered the performance of every team in every game throughout the competition. It was believed that due to slight differences in the interpretation of the coach language semantics, it was very difficult to support another team's coach.

What is also interesting is that, in addition to winning the competition with its coach OWL, the ChaMeleon-2001 appeared to be the most coachable team in the competition - the team with the largest overall goal differential. This raised many questions, but statistically significant results cannot be inferred from such a few number of games. It became obvious that in order to reach any conclusions about the effects a coach has on a team, many experiments would need to be conducted.

After RoboCup, hundreds of games were played using different coaches as well as different advice taking strategies in an attempt to better understand exactly what happened during the coach competition at RoboCup-2001 and determine exactly how much impact a coach has on a team. For each experiment, a different coach or advice-taking strategy was used during thirty regulation length games. The goal differentials were based on a separate set of games using no coach - the same as in the coach competition. The fixed opponent remained the same as in the coach competition, Gemini.

Figure 2 shows the resulting goal differentials using the ChaMeleons-2001 with each of the coaches from that entered the coach competition with the exception of Wright Eagle. Wright Eagle's coach was implemented in Windows and the testing environment set up could not support its inclusion. A statistically significant improvement in performance is shown by the ChaMeleons-2001 using every coach tested. Using its own coach, OWL, the ChaMeleons-2001 experienced a three times improvement over the baseline goal differential.

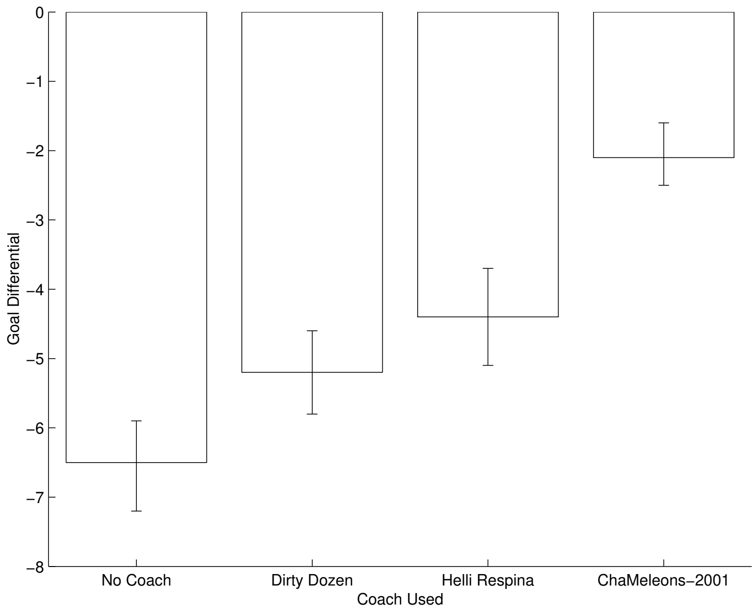

Fig. 2. Goal Differentials Using Several Coaches

Figure 3 shows the results of the experiments using different advice taking strategies with a single coach, OWL. The normal coach is the same coach used above (with the same results). The strict strategy blindly follows all advice that is given to the agents. As shown, it performs slightly worse though statistically significant at a level $p < .05$ with a two-tailed t-test. The random strategy is one in which truly random advice is sent to the agents. As expected, it performs far worse than any of the other strategies including not listening to the coach at all.

6 Conclusion

The action-selection architecture for the ChaMeleons-2001 simulated robotic soccer team was designed to integrate advice from many sources. This year, advice from only the coach agent was used. It was shown experimentally that the architecture is successful in its integration of advice and choosing when, and when not, to listen to the coach agent's advice. The team saw improvement while using every coach tested with. The success of the ChaMeleons is due to the agents

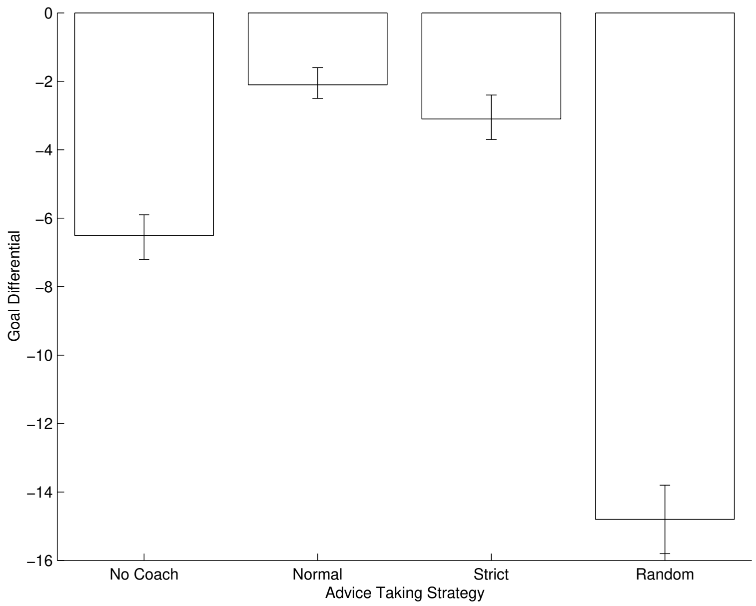

Fig. 3. Goal Differentials Using Different Advice-Taking Strategies

ability to ignore the coach when the agents believe they are able to execute a higher priority action. It can also be concluded that a coach agent does indeed have the potential to greatly impact the performance of a team.

Although it has been shown that the ChaMeleon's are indeed a highly coachable team, other possibilities for the architecture lie ahead. For example, it is possible for the behavior descriptors used by the *handleBall* to also describe the sequence of actions performed by an opponent. Given these sequences, rules can be inferred to describe when an opponent performs an action or an opponent's equivalent ordering of priority levels can be learned. A team receiving this advice from a coach could learn how to mimic the playing style of its opponents - hence the name ChaMeleons.

Another possibility for the ChaMeleons is the inclusion of several advice sources. Players could potentially receive advice from other players on the team, including a "captain." This raises many questions including which piece of advice does an agent follow when more than one rule matches from different sources or how to handle conflicting advice from multiple sources.

The action-selection architecture for the ChaMeleons-2001 was designed to facilitate both the integration of advice into the agents set of beliefs as well as other future learning tasks. An attempt was made to ensure that the architecture was not domain-dependent and could easily be adapted to other problems as well. As agent research shifts from learning to learning from other agents, the architecture proposed will provide a framework for others considering some of the issues involved with the integration of advice.

References

1. Ciprian Candea and Marius Staicu. AIRG Sibiu. In Peter Stone, Tucker Balch, and Gerhard Kraetszchmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, pages 409–412. Springer Verlag, Berlin, 2001.
2. Paul Carpenter, Patrick Riley, Gal Kaminka, Manuela Veloso, Ignacio Thayer, and Robert Wang. ChaMeleons-2001 team description. In *RoboCup-2000: Robot Soccer World Cup V*. Springer Verlag, Berlin, 2002.
3. Christian Drucker, Sebastian Hubner, Esko Schmidt, Ubbo Visser, and Hans-Georg Weland. Virtual werder. In Peter Stone, Tucker Balch, and Gerhard Kraetszchmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, pages 421–424. Springer Verlag, Berlin, 2001.
4. Tetsuya Esaki, Taku Sakushima, Shinji Futamase, Nobuhiro Ito, Tomoichi Takahashi, Wei Chen, and Koichi Wada. Kakitsubata team description. In Peter Stone, Tucker Balch, and Gerhard Kraetszchmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, pages 425–428. Springer Verlag, Berlin, 2001.
5. B. Grosof. Conflict handling in advice taking and instruction. Technical report, IBM Research Report 20123, 1995.
6. M.I. Jordan and R.a. Jacobs. Hierarchical mixture of experts and the em algorithm. In *Neural Computation*, 1993.
7. R. Maclin and J.W. Shavlik. Incorporating advice into agents that learn from reinforcements. In *Proc. of AAAI-1994*, pages 694–699, 1994.
8. Frederic Maire and Doug Taylor. A quadratic programming formulation of a moving ball interception and shooting behaviour, and its application to neural network control. In Peter Stone, Tucker Balch, and Gerhard Kraetszchmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, pages 327–332. Springer Verlag, Berlin, 2001.
9. J. McCarthy. Programs with common sense. In *Proc. Symp. on the Mechanization of Thought Processes*, pages 77–81, 1958.
10. Karen L. Myers. Planning with conflicting advice. In *Artificial Intelligence Planning Systems*, pages 355–362, 2000.
11. Itsuki Noda, Hitoshi Matsubara, Kazuo Hiraki, and Ian Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12:233–250, 1998.
12. Patrick Riley, Peter Stone, David McAllester, and Manuela Veloso. Att-cmunited-2000: Third place finisher in the robocup-2000 simulator league. In Peter Stone, Tucker Balch, and Gerhard Kraetszchmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, pages 489–492. Springer Verlag, Berlin, 2001.
13. Patrick Riley, Manuela Veloso, and Gal Kaminka. An empirical study of coaching. In *Distributed Autonomous Robotic Systems 6*. Springer-Verlag, 2002. (to appear).
14. Patrick Riley, Manuela Veloso, and Gal Kaminka. Towards any-team coaching in adversarial domains. In *AAMAS-02*, 2002. (poster paper) (to appear).
15. RoboCup Federation, http://sserver.sourceforge.net/. *Soccer Server Manual*, 2001.
16. Birgit Schappel and Frank Schulz. Mainz rolling brains 2000. In Peter Stone, Tucker Balch, and Gerhard Kraetszchmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, pages 497–500. Springer Verlag, Berlin, 2001.
17. Peter Stone, Patrick Riley, and Manuela Veloso. The cmunited-99 champion simulator team. In *RoboCup-99: Robot Soccer World Cup III*. Springer Verlag, Berlin, 2000.