Building Agent Teams Using an Explicit Teamwork Model and Learning

Milind Tambe, Jafar Adibi, Yaser Al-Onaizan, Ali Erdem
Gal A. Kaminka, Stacy C. Marsella, Ion Muslea

Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292, USA
robocup-sim@isi.edu
Tel: 310-822-1511

January 14, 1999

Abstract

Multi-agent collaboration or teamwork and learning are two critical research challenges in a large number of multi-agent applications. These research challenges are highlighted in RoboCup, an international project focused on robotic and synthetic soccer as a common testbed for research in multi-agent systems. This article describes our approach to address these challenges, based on a team of soccer-playing agents built for the simulation league of RoboCup — the most popular of the RoboCup leagues so far.

To address the challenge of teamwork, we investigate a novel approach based on the (re)use of a domain-independent, explicit model of teamwork, an explicitly represented hierarchy of team plans and goals, and a team organization hierarchy based on roles and role-relationships. This general approach to teamwork, shown to be applicable in other domains beyond RoboCup, both reduces development time and improves teamwork flexibility. We also demonstrate the application of *off-line* and *on-line* learning to improve and specialize agents' individual skills in RoboCup. These capabilities enabled our soccer-playing team, ISIS, to successfully participate in the first international RoboCup soccer tournament (RoboCup'97) held in Nagoya, Japan, in August 1997. ISIS won the third-place prize in over 30 teams that participated in the simulation league.

1 Introduction

Increasingly, multi-agent systems are being designed for a variety of complex, dynamic domains. Effective agent interactions in such domains raise some of most fundamental research challenges for agent-based systems. An agent in such domains may often need to model other agents' behaviors, or learn/adapt from its interactions, or form teams and act effectively in a team, or negotiate with other agents, and so on. For each of these research challenges, the uncertainty and the presence of multiple cooperative and non-cooperative agents, only conspires to exacerbate the difficulty.

To pursue such research challenges, the RoboCup research initiative has proposed simulation and robotic soccer as a common, unified testbed for AI research in general, and multi-agent research in particular[14, 13] (*www.robocup.org*). By focusing on a common, standard testbed, and highlighting several research issues of critical concern in multi-agents, the RoboCup effort aims to accelerate the advance of the state-of-the-art in multi-agent systems. Indeed, the RoboCup initiative has proved extremely popular with researchers, with annual competitions in several different leagues. Of particular interest in this article is the simulation league, which has attracted the largest number of participants. The stated research goals of the simulation league are to investigate the areas of (i) multi-agent teamwork, (ii) multi-agent learning, and (iii) agent modeling (plan-recognition)[15].

Yet, the lessons learned by researchers participating in RoboCup, particularly the simulation league, have largely not been reported in a form that would be accessible to the research community at large. There are just a few notable exceptions (e.g., [27]). However, extracting such general lessons in areas of teamwork, agent modeling and multi-agent learning, so as to be applicable in other domains beyond RoboCup, is an important task for RoboCup researchers.

This article attempts to remedy the above situation by focusing on two of the RoboCup research challenges — *teamwork* and *multi-agent learning* — and attempting to extract general lessons from our experience in building an agent-team for RoboCup. These two research challenges are of critical concern in a wide variety of multi-agent domains, ranging from virtual environments for training[33, 22], to multi-agent entertainment[9], to information-integration on the Internet[34], to multi-robotic space missions. With respect to teamwork, one key challenge is to enable an agent team to perform coherently in highly uncertain, dynamic environments. In particular, in such complex environments, individual team members often encounter differing, incomplete, and even inconsistent views of their environment. Furthermore, individual members can unexpectedly fail in fulfilling their responsibilities. To surmount such uncertainties and maintain coherence in teamwork, team members must be provided the capability of highly flexible coordination and communication. With respect to multi-agent learning, a key challenge is to enable individuals and teams to improve performance, despite the complexities introduced by the presence of other agents in the multi-agent environments.

This article presents our approach to address the above research challenges. For teamwork, we investigate the use of a novel approach, where agent-teams are developed based on an explicitly represented hierarchy of team goals and plans, an explicitly represented team organization hierarchy, and more importantly, a general-purpose, explicit model of teamwork[11, 32]. The teamwork model is an attempt to provide agents with commonsense knowledge of teamwork, particularly in terms of individuals' commitments and responsibilities in teamwork. There are two key implications of this approach to team development. First, it speeds up team development by providing both a conceptual framework and implementation support in terms of reusable code. Second, it promises to improve teamwork flexibility, since the teamwork model enables agents to autonomously reason about coordination and communication. Such autonomous reasoning contrasts with an approach that relies purely on pre-planned domain-specific coordination, which often leads to inflexibility.

Thus, one of the goals of this article is to understand the benefits and costs of pursuing the above teamwork-model-based approach, an issue highly relevant for developing agent-teams for a variety of multi-agent domains. To this end, we investigate the (re)use of STEAM[29, 31, 32] (a **Shell** for TEAMwork), in building agent-teams in RoboCup. STEAM is a state-of-the-art system, and it contributes in two ways in building RoboCup agent-teams. First, it provides a framework for rapid development of agent teams based on a hierarchical role-based team organization and a hierarchical description of team and individual actions, with coordination relationships among them. This framework is not only a conceptual one, rather, the STEAM implementation supports a concrete realization of this framework. Second, STEAM provides an explicit teamwork model, attempting to enhance teamwork flexibility, by providing agents abilities to autonomously reason about coordination and communication in teamwork. In STEAM, the teamwork model is based both on the *joint intentions* theory[3, 16] and the SharedPlans theory[7, 6, 8], but with key enhancements to reflect the constraints of real-world domains. The article discusses in detail STEAM's contribution in building RoboCup teams. It also discusses the difficulties we have faced in applying STEAM in RoboCup, and thus points to areas of future work.

With respect to agent learning, the article explores the effectiveness of learning in specializing and/or improving an individual agent's skill-set in the presence of multiple other agents in the environment. We have focused on a *divide-and-conquer* learning approach in designing agents. With this approach, skills for different modules within individual agents were learned separately, using different learning techniques. In particular, one of the skills, to pick a direction to shoot into the opponents' goal while avoiding opponents, was learned *off-line* using C4.5[21]. Another skill, to intercept the ball, used a mix of *off-line* and *on-line* reinforcement learning. One of the key surprises here was the degree to which individual agents specialized in their individual tasks in a given team. Thus, sharing experiences of different individuals in the team would in some cases appear to be significantly *detrimental* to team performance.

The vehicle for all of our research investigations in RoboCup is **ISIS** (**ISI S**ynthetic), a team of synthetic soccer-players that successfully participated in the simulation league of RoboCup'97. Developed at the University of Southern California's Information Sciences Institute (ISI), ISIS won the third place prize in the simulation league out of 32 teams.[1] ISIS was also the top US simulation team out of at least five US teams.

The rest of this article is organized as follows. Section 2 describes the RoboCup simulation domain and the architecture of an individual ISIS agent. This description is the basis for the discussion of the research issues. After an overview of STEAM in Section 3, Section 4

[1]Three out of the 32 teams forfeited, so that 29 agent-teams actually played.

discusses the application of STEAM in ISIS. Section 5 discusses learning in ISIS. Section 6 provides an evaluation of ISIS, while Section 7 discusses related work. Section 8 then provides a summary and topics for future work.

2 Background

This section first describes the RoboCup simulation domain in Section 2.1. Researchers familiar with the domain may skip this description. Next, Section 2.2 describes the architecture of individual ISIS agents.

2.1 RoboCup Domain Description

Computer chess has long been a standard problem for AI research. Chess is a turn taking, static game where players have complete and accurate information about the current status of the game with a centralized control. The recent success of the Deep Blue team in computer chess increases the need for a new research challenge which involves more complex tasks. Soccer is a dynamic, real time game with incomplete sensory information and distributed control. Robocup is an attempt to stimulate AI and robotics research based on robotics and synthetic (simulated) soccer as a common testbed [15, 14].

For readers unfamiliar with Soccer, here is our attempt to briefly describe the game. Soccer is a continuous game between two teams competing in a rectangular field, each attacking a goal at the end of the opposite side of the field. The borders of the soccer field are referred to as sidelines (or touch lines). A soccer match is played in two halves of equal time duration and two extra periods (in playoff games, if necessary). The team scoring a greater number of goals during the entire course of the match is the winner. Each team consists of eleven players, one of whom is a designated goal keeper (the goal keeper was not implemented in the Robocup'97 competition). A kick-off from the center of the field is the way for starting the match. The play also resumes from the center after any team's scoring a goal, starting the second half, or starting the extra time period. All players must be in their own side of the field before a kick-off.

Due to the dynamic nature of the game, a team plays according to a general team strategy but not to detailed step-by-step instructions. The ultimate goal of a team engaged in a soccer match is winning the match by scoring as many goals as possible and attempting to keep the opponents from scoring. In order to achieve the team's goal, soccer players are expected to possess several defensive and offensive skills (depending on their role in the game) with different levels of sophistication. Some of these skills are: passing to a teammate, shooting at the opponent's goal, dribbling, intercepting the ball, clearing the ball away from own goal area, marking opponents, tackling opponents, and positioning in the field. These skills often require a great deal of coordination between the player and one or more of his teammates. For example, a player passing the ball to a teammate expects the intended receiver to be prepared to intercept the ball in order for the pass to be successful.

This article focuses on the simulation league of RoboCup. This league uses a standard 2D simulation testbed called the *soccer server*. This standard server enables agents of two different teams to compete by playing a simulated soccer game. The server provides 11 player "bodies" per team, to be controlled by 11 separate agents. No centralized control is allowed. Furthermore, agent processes cannot directly communicate with one another; instead all such communication is mediated by the soccer server as discussed below. Figure 1 shows a snapshot of the soccer server with two competing teams: CMUnited [25] versus our *ISIS* team.

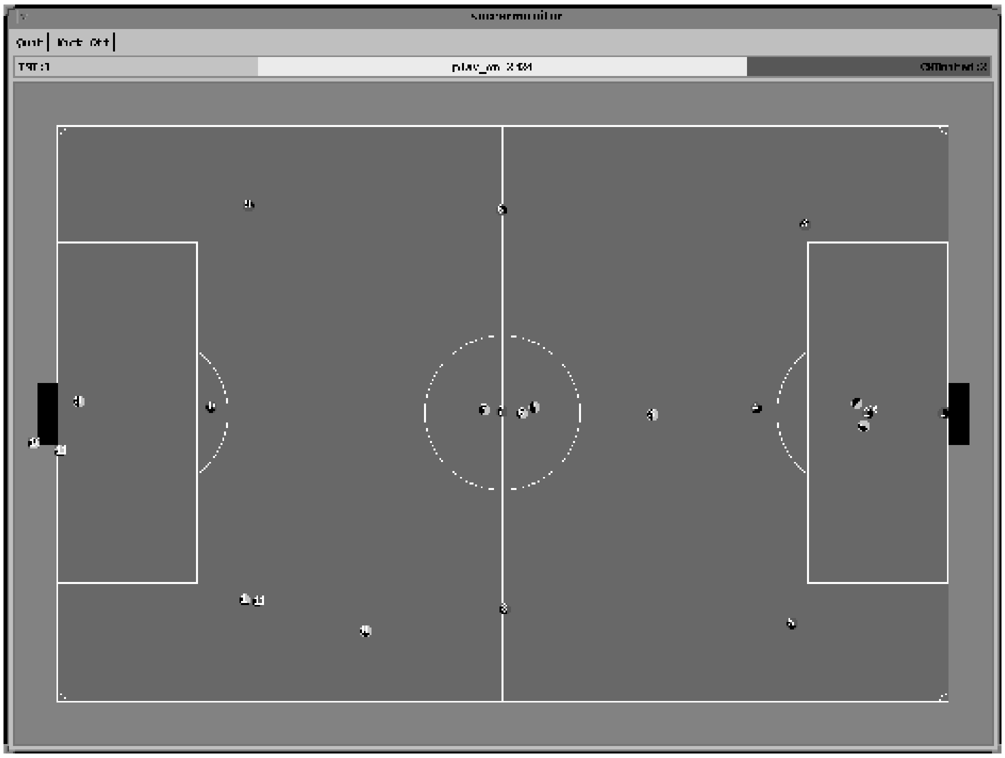

Figure 1: The Robocup synthetic soccer domain.

The soccer server provides each individual player agent input in the form of two types of sensory information: visual information (i.e. *see* messages) and audio information (i.e. *hear* messages). *See* messages are sent by the soccer server every 300 ms[2] and they provide information about the distance and angle to visible stationary landmarks (e.g. corner flags, center spot, and goal posts); team name, uniform number, distance and angle of players within a certain range based on the current setting of the agent's visual field; distance and angle to the sidelines (the field has 4 sidelines, however not all of them are visible at a given time); the distance, distance change rate, angle, and angle change rate to the ball. The stationary landmarks are distributed inside and outside the field. Since no direct information about an agent's own position is being provided by the server, the landmark information is very useful to help agents orient themselves in the field. Agents have a limited perceptual field based on the direction they are facing, and thus do not possess perfect information about their surroundings. *Hear* messages are either announcement from the server (e.g. corner kick call, or free kick call) or messages that have been posted by other players.

[2]While the rate was 300 ms in the RoboCup'97 competitions, more recently, the rate for See messages has changed to 150 ms.

With respect to output, player agents are equipped with the following set of primitive actions: *say* to post messages, *dash* with a certain power in the agents facing direction, *turn* to change the players facing direction, and *kick* to kick the ball with a certain power and direction. These primitive actions are used as building blocks to implement more sophisticated actions. For example, dribbling might involve a kick, a dash, and a turn. Agents are allowed kick, dash, or turn action every action period (in RoboCup'97, it was set to 100 ms).

The soccer server captures many of the complexities of real-world soccer, including the following:

- Players have limited perceptual information.
- Players can communicate only by posting messages through the soccer server which are accessible only by players within a certain range.
- Players have limited stamina which is consumed by dashing actions.
- Actions and sensory information sent to players are noisy.
- The game occurs in a real time and dynamic environment.

Work continues to improve the quality of the soccer server, and to continue to make it increasingly realistic.

2.2 ISIS Individual Agent Architecture

The architecture of an individual ISIS player agent as shown in figure 2, has two layers of abstractions. The lower layer, input and output processing, provides the interface between the soccer server and the agent. This layer does all the preprocessing of the sensory information sent by the soccer server and prepares them for the use by the higher layer. The higher layer handles all the decision making for the agent using the information provided by the lower layer.

The lower layer consists of two components, input and output processing. Both components use a shared communication library to send and receive messages to/from the soccer server. The input processing module first decides which messages to process and which messages to drop if there is an overflow in the incoming message buffer. Generally, *hear* messages are considered more important than the *see* messages and are never dropped. If there is more than one *see* message waiting, only the last *see* message is processed and all the prior *see* messages are dropped.

Then, the received messages are parsed and the data structures that correspond to the state of the game are updated. The state of the game for the last few cycles is remembered and this information is used when necessary. For example, if the ball is not visible in the current cycle, the last known position of the ball is used.

After the messages are parsed and the data structures are updated, the input processing module calls separate subroutines for calculating the parameters of interest for the higher layer. The results of these calculations are communicated to the higher layer using the shared variables. These parameters can be categorized in two major groups:

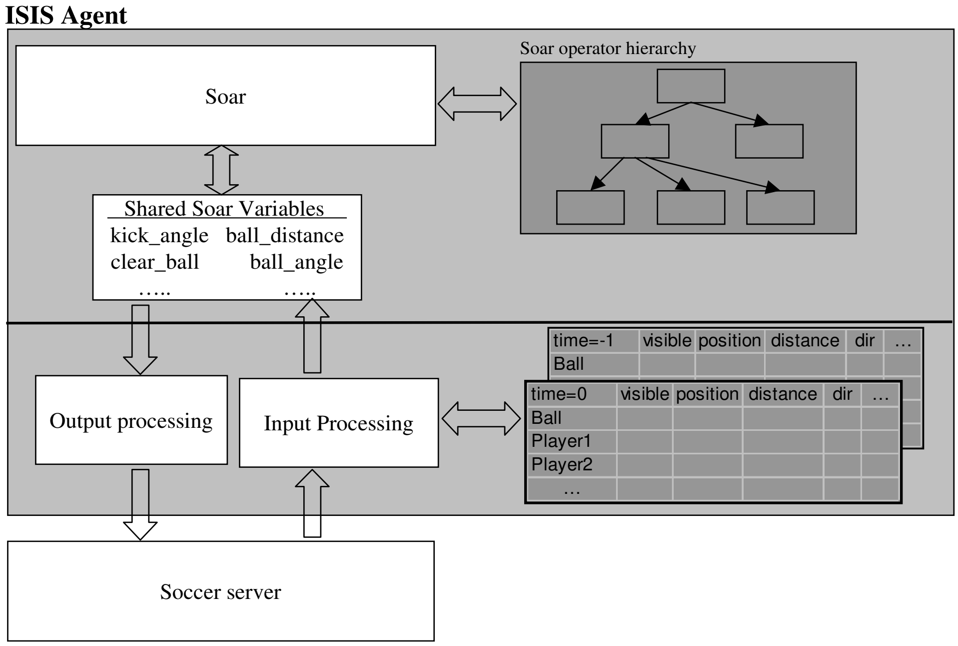

Figure 2: ISIS agent architecture

- *Kicking directions:* These parameters contain the recommended direction to kick the ball. Three possible directions to kick the ball are calculated: (i) a group of C4.5 rules compute a direction to intelligently shoot the ball into the opponents' goal (discussed further in Section 5); (ii) a hand-coded routine computes kicking direction to clear the ball; (iii) a second hand-coded routine computes direction to kick the ball directly into the center of the opponent's goal (without taking opponents' location into account).

- *Interception parameters:* Interception subroutines compute a plan consisting of turn or dash actions for the player if the player is to intercept an approaching ball (also discussed further in Section 5).

The lower layer does not make any decisions with respect to its recommendations however. For example, it does not decide which one of its three suggested kicking directions should actually be used by a player-agent. Instead, all such decision-making rests with the higher layer implemented in the Soar integrated AI architecture[20, 24]. Once the Soar-based higher layer reaches a decision, it communicates with the lower layers, which then send the relevant commands to the soccer server.

We found the timing of the command submissions to the server to be very critical in agent's performance, since the soccer server will drop the extra commands submitted in an action cycle. To prevent this, the output processing module in the lower layer provides synchronization between the soccer server and the agent.

The higher layer, Soar architecture, involves dynamic execution of an operator (reactive plan) hierarchy. An operator begins execution (i.e., the operator is activated), when it is selected for execution, and it remains active until explicitly terminated. Execution of high-level abstract operators leads to subgoals, where new operators are selected for execution, and thus a hierarchical expansion of operators ensues. An operator is composed of three sets of rules: (i) precondition rules; (ii) application rules; and (iii) termination rules. Precondition rules help select operators for execution based on the agent's current high-level goals and beliefs about its environment. Once operators are selected for execution, they are executed by the application rules. If the agent's current beliefs match an operator's termination rules, then the operator terminates. Agents built in other architectures such as PRS[10], BB1[9], RAP[5] for dynamic domains may be similarly characterized in this fashion.

The key here is that in a dynamic domain such as Robocup, the operator selection, execution and termination may be heavily dependent on the current state of the environment as perceived by the individual agent. For instance, figure 3 illustrates a portion of the operator hierarchy for ISIS player-agents in RoboCup.[3] At any one time, depending on the state of the environment, only one path through this hierarchy is selected or active. Thus, an agent may have selected the *intercept* operator (under *careful-defense*) to intercept the ball based on its current perception of the environment. However, the agent will terminate *intercept* and switch to the *locate-invisible-ball* operator if the ball becomes invisible unexpectedly before the completion of the intercept.

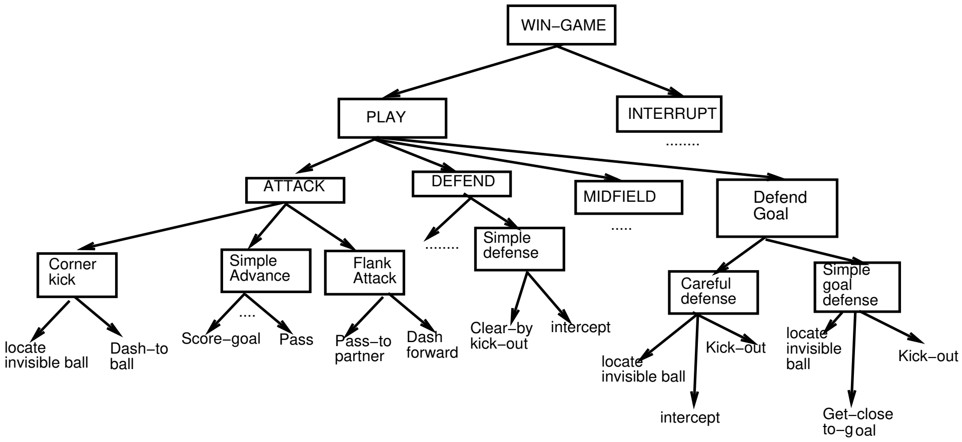

Figure 3: A portion of the operator hierarchy for player-agents in RoboCup soccer simulation. Boxed operators are team operators (discussed in a following section), others are individual operators.

3 Overview of STEAM

As mentioned earlier, to address the teamwork challenge, we explore a new teamwork-model based approach for agent-team development, via the the application of the STEAM system in RoboCup. STEAM was originally developed in the context of building teams of attack-helicopter pilot-agents for real-world military simulations[32, 33]. STEAM was later reused in the same simulation environment, for building teams of transport helicopter pilot agents. RoboCup is a third domain for STEAM's reuse. It provides a challenging domain for testing

[3]Some of the operators shown were encoded after the RoboCup'97 competition.

the generality and reuse of STEAM concepts and code, given that this is a substantially dissimilar domain. Indeed, some of the generalizations in STEAM were inspired by its application in RoboCup. This section provides only a brief overview of STEAM. For a more extensive discussion of STEAM please see [32].

There are two key aspects of STEAM that are relevant in building agent-teams. First, it provides an implemented framework for team development, based on two separate hierarchies, each with key constraints:

- *Team organization hierarchy and roles:* In STEAM, a team may have a flat or a hierarchical organization (where a team may be recursively composed of subteams). Thus, for instance, a *company* of attack helicopter pilots (8 helicopters) may be composed of two *teams*, which in turn consist of individuals. A critical feature of this organization hierarchy is that it may be based on particular *roles*. Roles are of two types:

  Persistent roles: These are long-term assignments of roles to the individuals or subteams in the organization. For instance, in the synthetic attack-helicopter domain, a team may be designated as the scouting team, or an individual may be designated the commander. This assignment typically will not change in the short term.

  Task-specific roles: These are shorter-term assignments of roles, based on the current task and situation. For instance, in formation flying in the attack-helicopter domain, the assignment of the leader of the formation is determined based on the type of formation. For instance, a particular helicopter *H1* may be the leader when flying into the battlefield, but another helicopter *H2* may be the leader of the formation when flying out of the battlefield.

  There are specific properties associated with each team, subteam or individual. For instance, teams have a set of members, they may possess *capabilities*, and may have one or more communication channels with specific associated communication commands and costs. The assignment of roles to individuals or subteams is based on their capabilities. However, this assignment may not be provided ahead of time, so that individuals may need to volunteer or be requested to fill in the roles. Furthermore, if an individual's or subteam's capability degrades during performance, roles may be reassigned (as discussed in more detail later in this section).

- *Team activity hierarchy:* STEAM relies on an explicit representation of team activities in the form of *team operators* or reactive team plans. Team operators explicitly express a team's joint activities, unlike the regular "individual operators" which express an agent's own activities. For instance, in Figure 3, boxed operators such as $\boxed{\text{WIN-GAME}}$ are explicitly represented team operators. When all of the player agents in a team select the $\boxed{\text{WIN-GAME}}$ operator in their operator hierarchy, the $\boxed{\text{WIN-GAME}}$ team operator is considered selected by the team for execution. There is a hierarchy of such team operators, which bottom out in individual operators (not boxed). As with individual operators, team operators also consist of: (i) precondition rules; (ii) application rules; and (iii) termination rules. However, while an individual operator applies to an agent's private state (an agent's private beliefs), a team operator applies to an agent's *team state*. A team state is the agent's (abstract) model of the team's mutual beliefs about the world, e.g., the team's currently mutually believed strategy. There is of course no shared memory, and thus each team member maintains its own copy of the team state, and any subteam states for subteams it participates in. To preserve the consistency of a (sub)team state, a (sub)team member modifies its copy of the (sub)team state only through the use of (sub)team operators.

The mapping between the two hierarchies mentioned above is centered on roles. In particular, suppose a team is executing a team operator, with several suboperators. Then, the role of a subteam or individual in that team may constrain the suboperators it can execute in service of the team operator. Thus, if a team Θ is executing a team operator $\boxed{\text{OP}}$, then the assignment of a *role* γ_i to a member μ of Θ, constrains μ to a certain subset σ_i of the suboperators of $\boxed{\text{OP}}$. Furthermore, roles may have specific *coordination relationships* among them. For instance, the execution of operators for one role may be dependent on operators for another role. The concept of roles has been discussed before in the multi-agent literature[12]. The key in STEAM is the instantiation of this concept as part of the team organization hierarcy, with its use as a constraint on the selection of the sub-operators of a team operator, and its use in expressing coordination relationships.

The second, and perhaps the more important contribution of STEAM is that it provides an explicit, general-purpose teamwork model, which encode domain-independent axioms and theorems that explicitly outline team members' responsibilities and commitments in teamwork. In essence, it attempts to provide agents with commonsense knowledge of teamwork, that would enable them to autonomously reason about coordination and communication in teamwork. Thus, it aims to substantially reduce the encoding effort, and improve teamwork flexibility. STEAM uses the joint intentions theory[16, 3] as the basic building block of teamwork, but it is also strongly influenced by the SharedPlans theory[7, 8], and constraints realized in practical applications.

The tie between the teamwork model and the previously discussed team operators is as follows: when executing (sub)team operators, agents bring to bear all of the teamwork reasoning of a general teamwork model, which facilitates their communication and coordination. Currently, STEAM's teamwork knowledge could be categorized into three classes of domain-independent teamwork axioms. The first class is *coherence preserving* or *CP* axioms. These require that team members communicate with others to ensure coherent initiation and termination of team operators. Thus, for instance, if an agent privately discovers that the currently selected team operator is unachievable, then it must not just terminate the team operator on its own. Instead, it must communicate this unachievability information about the team operator to its teammates, so that teammates do not waste their resources, and that the team as a whole coherently terminates the team operator. The second class of *monitor and repair* or *MR* axioms detect if a team task is unachievable due to unexpected member (individual or subteam) failure. It then leads the team into reorganization — via reassignment of roles — to overcome this failure. As an example of STEAM's application, consider the synthetic attack-helicopter pilot team in a situation where the pilot team is flying in formation. If one of the helicopter pilots sees some unanticipated enemy vehicles on their flight route, STEAM's *CP* axioms require that this pilot inform its teammates about the enemy units (since it makes the relevant team operator unachievable), so that the team reacts coherently to the enemy.

The third set of *selectivity-in-communication* or *SC* axioms in STEAM arise because communication to attain perfect coherence in teamwork can sometimes be excessive, and indeed it may have highly detrimental side-effects[32]. Hence, STEAM also includes decision-theoretic communication selectivity. The key idea is to explicitly reason about the costs and benefits of different techniques for attaining mutual beliefs. For instance, in some cases, if there is high likelihood that the relevant information can be obtained by other teammates via observation (even if with a slight delay), then costly communication can be avoided. In contrast, communication becomes essential if there is low likelihood that teammates can obtain the relevant information independently, and if there is a high cost to not making this information available to the teammates. For this decision-theoretic reasoning, one of the critical questions is determining the costs associated with the communication channels available the team. Such costs may indeed change over time. The next section discusses some experiments, trying to determine if the communication costs were reasonably accurately set in ISIS during the RoboCup'97 competitions.

STEAM's teamwork model has also been currently encoded in the Soar integrated architecture, in the form of 283 rules[32]. These rules, with documentation, traces and pointers to their usage are available at (*www.isi.edu/teamcore/tambe/steam/steam.html*).

4 Applying STEAM in ISIS

The application of STEAM in ISIS tests a novel approach to building agent-teams. This section discusses STEAM's application in ISIS in further detail. Section 4.1 describes the team organization and the team operator hierarchies. Section 4.2 discusses the use of the teamwork model in ISIS. Section 4.3 evaluates STEAM in ISIS.

4.1 Team Organization and Team activity Hierarchies

Figure 4 shows the team organization hierarchy in ISIS. The top-level node represents the entire team. This team is currently divided into four subteams, each with a different *role*: (i) forwards; (ii)midfielders; (iii) defenders; (iv) goalies. There are two-three roles per subteam as shown in the figure. For instance, agents in the subteam of forwards have roles such as "center", "left" and "right". More than one agent can play the same role. For instance, it is possible to have two centers in the forward subteam.

Figure 3 shows the team operator hierarchy for ISIS agents. As discussed before, operators shown in boxes such as $\boxed{\text{WIN-GAME}}$ are team operators, while others are individual operators. $\boxed{\text{WIN-GAME}}$ is the highest-level team operator. When all of the player agents select $\boxed{\text{WIN-GAME}}$ in their operator hierarchy, the $\boxed{\text{WIN-GAME}}$ team operator is established. There are two possible operators that can be executed in service of $\boxed{\text{WIN-GAME}}$, specifically $\boxed{\text{PLAY}}$ (when the ball is in play) or $\boxed{\text{INTERRUPT}}$ (when the ball is not in play, and the players may be attempting to take up positions on the field). The execution of any of $\boxed{\text{WIN-GAME}}$, $\boxed{\text{PLAY}}$ or $\boxed{\text{INTERRUPT}}$ involves the entire team.

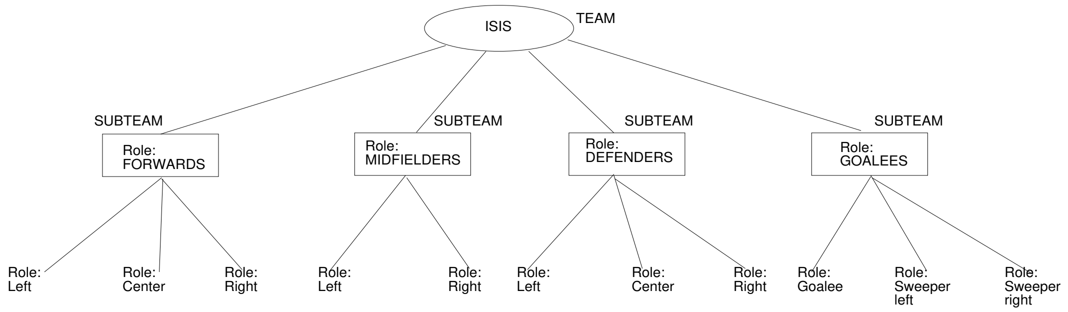

Figure 4: The team organization hierarchy in ISIS.

There are some team operators that are executed by subteams. Thus, once the $\boxed{\text{PLAY}}$ team operator is selected for execution, an agent selects one of its four children — team operators $\boxed{\text{ATTACK}}$, $\boxed{\text{MIDFIELD}}$, $\boxed{\text{DEFEND}}$ and $\boxed{\text{DEFEND-GOAL}}$ — depending on the role of the subteam the agent participates in. Thus, if an agent participates in a subteam with the role of *forwards*, then it executes the $\boxed{\text{ATTACK}}$ team operator. In service of $\boxed{\text{ATTACK}}$, the entire subteam may execute $\boxed{\text{FLANK-ATTACK}}$, $\boxed{\text{CORNER-KICK}}$ or $\boxed{\text{SIMPLE-ADVANCE}}$. The roles of individuals within the *forwards* subteam may constrain the individual operators they execute in service of these team operators, although some operators could be executed by individuals in any of the roles. For instance, the individual operator "score-goal" may be executed by any of the agents in the forward subteam, to take advantage of any goal scoring opportunities.

4.2 Applying STEAM's Teamwork Model in ISIS

In applying STEAM's teamwork model, the *CP* axioms were relatively easily transferred to the RoboCup domain, to enable agents to communicate and ensure coherence in execution of team operators. A typical example of such communication is seen when three players in the "goalie" subteam execute the $\boxed{\text{DEFEND-GOAL}}$ team operator. In service of $\boxed{\text{DEFEND-GOAL}}$, players in this subteam normally execute the $\boxed{\text{SIMPLE-DEFENSE}}$ team operator to position themselves properly on the field and to try to be aware of the ball position. Of course, each player can only see in its limited cone of vision, and particularly while re-positioning itself, can be unaware of the approaching ball. Here is where CP axioms can be beneficial. In particular, if any one of the players in the goalie subtam sees the ball as being close, it declares to everyone in its subteam that the $\boxed{\text{SIMPLE-DEFENSE}}$ team operator is irrelevant. Its teammates now focus on defending the goal in a coordinated manner via the $\boxed{\text{CAREFUL-DEFENSE}}$ team operator. Should any one player in the goalie subteam see the ball move sufficiently far away, it again alerts its team mates (that $\boxed{\text{CAREFUL-DEFENSE}}$ is achieved). The subteam players once again execute $\boxed{\text{SIMPLE-DEFENSE}}$ to attempt to position themselves close to the goal. In this way, agents attempt to coordinate their defense of the goal.

Agents also jointly initiate team operators. For instance, when agents attempt a corner kick ($\boxed{\text{CORNER-KICK}}$ team operator), they ensure coherent initiation by exchanging messages. The result is that team members are appropriately positioned when a corner-kick is executed, greatly increasing the chances of scoring a goal.

Indeed, *all* communication in ISIS agents is currently driven by STEAM's general-purpose teamwork reasoning. Yet, given STEAM's decision-theoretic communication selectivity (*SC* axioms), not all team operators lead to communication in their initiation and termination. For instance, the INTERRUPT team operator is terminated by a referee's message to "kick-off". Since agents assume that all team members can hear a referee's message, they do not communicate in addition when terminating INTERRUPT. Such selectivity ensures a significant reduction in communication overheads.

There are unfortunately significant portions of STEAM that have yet to be applied in ISIS. These portions focus on *MR* axioms: reasoning about maintenance and repair of failed team operators, particularly via exchange of roles in teamwork. For instance, STEAM could potentially enable forwards to reason about aiding defenders if own goal is under threat. However, such reasoning requires some domain-dependent reasoning, possibly based on plan recognition, to first recognize the failures of own team members in fulfilling allocated roles. For instance, a defender may have rushed off to a side and may be unable to move back quickly enough to fulfill its role to defend its allocated area. In the absence of capabilities to recognize such role failures on part of individuals or subteams, STEAM's further application is currently hampered.

4.3 Evaluating STEAM in ISIS

There are several aspects of evaluating the use of STEAM in ISIS. One key aspect is its benefit in building an agent team. Here, STEAM's implemented framework was beneficial in developing the team organization and team operator hierarchies, and in providing readymade teamwork reasoning via the reuse of the teamwork model. We may approximately measure the reuse of the teamwork model in terms of the number of rules reused. This measure shows that the reuse has varied over time. At the time of RoboCup'97, about 101 rules, i.e., about 35% of STEAM rules, were reused. At the time of submitting this article, the number of rules used have increased to about 130 (which is roughly 45% of STEAM rules).[4] The remaining rules in STEAM could potentially be used, but as discussed in Section 4.2, further work is necessary to accomplish this use. Nonetheless, the existing level of reuse is still useful, since without this code, all of the communication for jointly initiating and terminating the key team operators would have had to be implemented via dozens of domain-specific coordination plans.

A second key aspect of the evaluation is impact of STEAM on ISIS performance in actual games. Here the aspect of STEAM that could be meaningfully evaluated is the contribution of the teamwork reasoning based on the teamwork model. To this end, we experimented with different settings of communication cost in STEAM. While in complex environments, the communication costs would vary over time, in RoboCup'97 competitions, we fixed the cost to "low" throughout the entire competition. At this "low" communication cost, ISIS agents communicate a significant number of messages (even though they do filter out several

[4] In the ISIS team fielded at RoboCup'98, interactions with the player resources (stamina) caused the reuse to go down to 35%.

messages that they would communicate if the communication cost were set to zero). At "high" communication cost, ISIS agents communicate no messages. Since the portion of the teamwork model in use in ISIS is effective only with communication, a "high" setting of communication cost essentially nullifies the effect of the teamwork model.

Tables below shows the results of games for the two settings of communication cost. These games were run on a soccer server version that was developed after the RoboCup'97 competitions (we discuss below the results on the server used at the RoboCup'97 competitions). Table 1 compares the performance of the two settings against the Andhill97 team developed by T. Andou of NTT labs, which came in 2nd in the RoboCup'97 competitions. Table 2 compares the performance of the two settings against the CMUnited97 team developed by Stone and Veloso of Carnegie Mellon University, which came in 4th in the RoboCup'97 competitions. Table 1 shows that with the "low" setting, the goal difference between ISIS and Andhill97 was -52 at 15 games (mean -3.46 per game), while with "high" settings, the goal difference was -66 (mean -4.40 per game) in 15 games. In contrast, Table 2 shows that with the "low" setting, the goal difference between ISIS and CMUnited97 was 49 at 15 games (mean 3.26), while with "high" settings, the goal difference was 26 in 15 games (mean 1.73). That is, communication with low costs helped to significantly improve ISIS's performance in both cases. Indeed, in both the following cases, the difference in the means of the goal-differences were seen to be significant under a T-test.

Communication cost	Total games	Total score in goals	Total Goal difference	Mean goal difference
Low	15	4-56	-52	-3.46
High	15	0-66	-66	-4.40

Table 1: The contribution of STEAM's communication to ISIS's performance when playing against Andhill97.

Communication cost	Total games	Total score in goals	Total Goal difference	Mean goal difference
Low	15	50-1	+49	3.26
High	15	31-5	+26	1.73

Table 2: The contribution of STEAM's communication to ISIS's performance when playing against CMUnited97.

The above results indicate that the communication instigated by the STEAM teamwork model did make a useful contribution to ISIS's performance. This does not imply that the "low" setting for communication costs in STEAM is the right one under all circumstances.

Indeed, in other instances, “high” might outperform “low”, so that no communication is the best strategy. For instance, identical experiments were also done with the soccer server version used during the RoboCup’97 competitions. The results there have shown a very large variance in the performance of the teams with “high” and “low” settings, so that the results cannot be reported meaningfully. At this stage, we can only speculate that the variance appears related to the amount of general network traffic and computer load. Thus, in that version of the soccer server, it is unclear at this stage if a particular communication cost setting is always the best, or if it should be varied over time, depending on the network load.

5 Agent Learning

Our work on agent learning in ISIS was inspired by previous work on machine learning in RoboCup[25, 19]. We focused on a *divide-and-conquer* learning approach in designing agents. With this approach, different modules (skills) within individual agents were learned separately, using different learning techniques. This approach was possible because of the separation of the individuals skills of the ISIS players, such as skills to kick in different ways, intercept the ball, pass etc. into different modules. There are two separate individual agent skills where learning has currently been applied in ISIS: (i) selecting an intelligent direction to shoot the ball when attempting to score a goal (using C4.5); (ii) selecting a plan to intercept a ball (using reinforcement learning).

5.1 Learning to Shoot to Score a Goal

Shooting a ball to score a goal is clearly one of the critical skills in soccer. Yet, our initial hand-coded approaches, based on heuristics such as ”shoot at the center of the goal”, or ”shoot to a corner of the goal”, failed drastically because of three main problems. First, these simple heuristics, which did not take into account the players’ configuration in the proximity of the goal, were often foiled even by stationary opponents. Second, as we attempted to introduce more complex heuristics, we discovered that for a given configuration of players around the opponent’s goal, a small variation in the shooter’s position may have dramatic effects on the best shooting direction. Third, a careful analysis of different game situations appeared to lead to a large number of heuristic rules.

To address these problems, we decided to rely on automated, *off-line* learning of the shooting rules: a human specialist created a set of 3000 shooting situations and selected the optimal shooting direction for each such situation; then we used these labeled shooting scenarios as training examples for a learning system. The choice of the learning system was a crucial decision, and we selected C4.5[21] for several reasons. First, C4.5 offers the appropriate expressive power because each game situation can be easily described in propositional logic. Second, considering the large number of shooting scenarios and the fact that in most of them only a fraction of the players were visible to the shooter, it was essential to choose a learning algorithm than can easily handle both missing attributes and a large number of training cases. Third, we needed a learning system that can generate sophisticated rules (such as C4.5’s decision trees) because our experiments showed that simplistic rules ultimately lead to a poor scoring capabilities. Last but not least, the real-time constraints of the soccer-server required a fast, C-written classifier.

The expert labeled each shooting scenario with one of the UP, DOWN, and CENTER kicking directions; when the shooting direction wasn't straightforward, the expert had the shooter execute all three kicks and selected the one with the best outcome. Each such labeled shooting scenario was used as a training case described by 40 attributes: the recommended kicking direction, the shooter's facing direction, and the shooter's angles to the other visible players, the 12 flags, the 4 lines, the ball, and the opponent's goal. Our decision to use only the angle information and to ignore the corresponding distances was motivated by the following reason. The purpose of recommending a kicking direction is to inform the higher-level (Soar) about the clearest path towards one of the three goal regions. As the higher-level is the one that selects the next action (e.g., shoot, pass, or clear the ball), it is its responsibility to take into account the distances between the ball and the opponents in order to estimate the chances of the ball being intercepted.

The system was trained on 1600 randomly chosen training cases, and the other 1400 examples were used for testing. We repeated this procedure 50 times, and the average accuracy of the rules on the testing sets was 70.8%. Even though the predictive power seems to be fairly low, the kicking rules were quite efficient in practice. This apparent paradox has several explanations. First, the shooting scenarios were quite exhaustive and included a significant number of shootings from far away and from "closed angles". In such situations, the difference between the three possible kicking directions are smaller than the noise and randomness of the Robocup domain. Second, as the higher-level typically selects the learned shooting direction only when players are reasonably close to the goal and can see the goal, it follows that the actual kicking directions is rarely used from unfavorable situations such as the ones described above.

While we had earlier started out with the three shooting directions only as a first step, after the extensive testing of the system it became clear that a finer discretization of the goal is unlikely to be beneficial because of two reasons. First, the relatively small dimensions of the goal in conjunction with the noise and randomness of the Robocup domain are not appropriate for finer kicking directions. Second, even with only three kicking directions, the human expert seemed to have trouble to be consistent in the labeling of the shooting scenarios. For instance, the average accuracy of the C4.5 rules over the 50 *training sets* was only 76.3%. Besides the noise and randomness of the domain, we found that the low accuracy over the training set was caused both by inconsistent labelings and by insufficient training data for some scenarios, which lead to the pruning of some important branches of the decision tree.

5.2 Learning to Intercept

Intercepting the ball is also a critical skill but it is not a simple skill. Whether in human soccer or RoboCup, there are many external and internal playing conditions that can impact a player's intercept. The opposing side may kick/pass/run harder than normal, thereby requiring a player to modify the path they take or forego interception. Changes in the environment that impact the ball's motion or visibility may also dramatically impact play. Human players, at least, fluidly adapt to these conditions. The same could not be said for the ISIS players fielded at the RoboCup'97 competitions.

The intercept used by ISIS at RoboCup'97 was hand-coded based on offline experiments and then fitted into the players. Our experience during RoboCup'97 and in subsequent testing has been that there is considerable variation as to how well these plans worked under actual playing conditions. The result seemed to depend on factors, some of which were external to the agent and could not be taken into account during the offline engineering of the intercept, such as reliability of perceptual updates, network and cpu load and the style of opponent play. What was certain that unlike real soccer players, our ISIS players' intercept skills were not adapting very well to differing internal and external factors.

One could approach this problem by engineering an intercept based on the details of how the soccer server calculates ball and player velocities, decays in these velocities, perceptual information sent to players, and the various randomizations of motion and perception. All these internal simulation factors are available to the designer. Using this information, one could, for example, build an intercept that simulated the soccer simulation in order to predict future states of the world based on the assumption that there are no other agents in the world.

We have taken a different approach, driven by the question as to what would happen if players could learn their intercept plans themselves and what would that learning tell us. To that end, we are exploring a reinforcement learning[28, 4] approach whereby players can adapt their intercept online, under actual playing conditions using just the perceptual input provided by the RoboCup'97 server to the player: *the ball's current direction, change in direction, distance.*

Although our focus is more on what is learned as opposed to how it is learned, any approach must nevertheless address the difficulties inherent in online learning of intercept. In the course of a game, there are not many opportunities to intercept the ball. Yet even one failed intercept can have dire consequences. Therefore, rapid learning of intercepts is critical. Indeed, if an agent waited for its intercept to complete in success or failure and then adapted its behavior based on the feedback only at the end of such an intercept, the result would appear to be inadequate for rapid, real-time adaptation. At the same time, since exogenous events may cause the intercept to fail, the system's need to fully exploit the learning opportunities is further accentuated.

In keeping with these concerns, we have pursued an approach whereby reinforcement occurs over a fixed temporal extent (a fixed horizon of actions), defined by an intercept *micro-plan*, and there is immediate reinforcement, occurring as this micro-plan is in progress. Specifically, the learning uses the same simple plan structure used in the non-learning version of the team, whereby a player intercepts the ball by stringing together a collection of micro-plans. For every step in a micro-plan, ISIS has an expectation as to what any new information from the server should inform it as to the ball's location. Failure to meet that expectation results in a learning opportunity. The micro-plans consist of a turn increment followed by one or two dashes. For most input conditions, the actual turn is calculated from the turn increment in the following fashion:

$$ActualTurn = CurrentBallDir + (TurnIncrement * ChangeBallDir)$$

Furthermore, learning is generalized spatially via a clustering of perceptual input conditions (i.e., states). For each cluster, the learning proceeds independently so, for instance, a different micro-plan may be learned for balls that are moving towards the player as opposed to balls moving away. Repeated failures lead to changes in the plan assigned to that input condition. In particular, the turn increment specific to that input condition is adjusted either up or down upon repeated failure. Currently, this clustering is designed by hand and is fixed, but automated, dynamic approaches may be applicable here (e.g., [18]).

Finally, to provide for a more accurate assessment of success or failure, evaluation itself is driven by an "oracle", ISIS's higher-level, decision-making tier, which tends to have more complete information on why an intercept plan is failing. For instance, failure due to a blocking player is treated differently from failure due to an improper turn angle.

5.2.1 Experimental Results

To date, we have performed several preliminary experiments in which ISIS made online adjustments to the turn increment factor. These experiments involved six extended-length games between ISIS and two other teams, CMUnited97 (team of Stone and Veloso of Carnegie Mellon) and Andhill97 (team of T. Andou of NTT labs). In each game, every ISIS player started with a default value of 2.0 for their turn increment across all input conditions.

By the end of game, learning can result in turn increment values that range from +5 down to -1, across input condition clusters. Because of the multiplicative factors and because the intercept plans are invoked repeatedly, these can constitute a dramatic change. However, for any particular input condition, the trend in the learning seems uniform in its direction across teams and positions. If one player playing Andhill97 increased the turn increment under a certain input condition then all the players with sufficient training examples would tend show an increase whether in games with Andhill97 or CMUnited97 for that input condition. There however were often striking differences in magnitude across teams and players. Below, we consider two illustrative examples.

Test 1: Same player against different Teams

In particular, lets consider the case of what a player in a particular position learns while playing a game against CMUnited97 as opposed to what the same player learns playing against Andhill97. In Figure 5, the mean results for Player 1, a forward, are graphed against the mean for all players, at 3000 ticks of the game clock until the end of the game at 15000 (RoboCup games normally run to 6000, here the time has been lengthened by a factor of 2.5 to simplify data collection). This particular data is for the input condition of balls moving across the player's field of vision, a middling-to-close distance away. Note that in games against Andhill97, the player is learning a turn increment similar to the mean across all players for this input condition. However, in games against CMUnited97, the player is learning a much larger increment. The difference for CMUnited97 and Andhill97 at 15000 ticks is significant (at a 5% level of significance using a t-test).

Test 2: Different Players Against Same Team

Different players against the same team can also learn different increments. Consider Figure 6 which depicts the same input condition for Player 10, a fullback. Against both Andhill97 and CMUnited97, the mean stabilizes around 3, in either case a significant difference with respect to the means seen for Player 1 against CMUnited97. In fact, to date, large values (greater than 4) have only been common in CMUnited97 games and only with respect

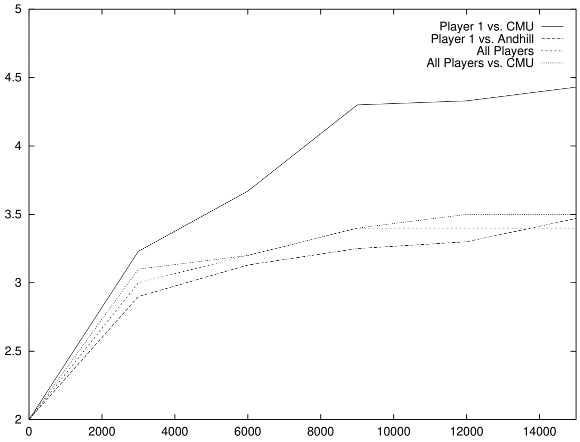

Figure 5: Contrast of Player 1 versus CMUnited97 with Player 1 versus Andhill97 under same input condition. Means across all players provided for comparison. Player 1 is a forward. (X-axis plots the simulation time, y-axis plots the turn increment.)

to Player 1 and to a lesser extent the other two forwards. However, similar values occur for at least two separate input conditions (the one discussed for Test 1, and a different input condition where the ball has the same sideways motion but is a little bit further away).

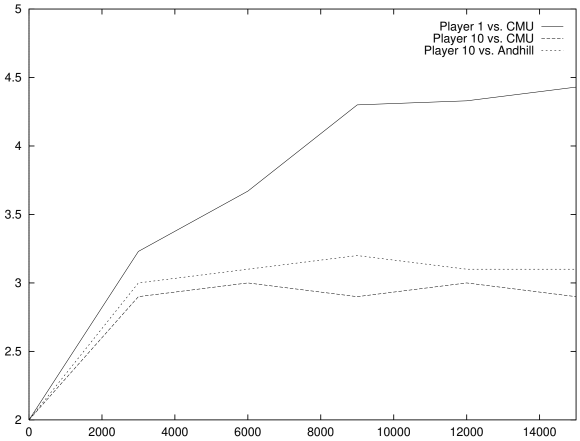

Figure 6: Contrast of Player 1 versus CMUnited97 with Player 10 versus both CMUnited97 and Andhill97, under the same input condition. Player 1 is a forward and Player 10 is a goal defender.

5.2.2 Analysis

In performing these experiments, we wanted to find out what ISIS would tell us about ball interception within the dynamics of the multi-agent RoboCup environment. We were not expecting so many revelations. As Test 1 and 2 reveal, Player 1 distinctly tailors its intercept to its role and its particular opponents. There is a domain-level analysis of these results, which clarified why player 1 had specialized its role so significantly — CMUnited97's defenders often cleared the ball with a strong sideways kick, which player 1 (and also other forwards) continuously faced. This maneuver catches non-learning ISIS players offguard, with their resulting turns and pursuit consistently lagging the ball's travel. ISIS's low-level learning is trying to compensate by turning more in order to cut the ball off. Thus, the learning in Player 1 was adapting to other agents via the unique interactions with them as the two sides respectively performed their roles.

These results reveal a significant role-based specialization of skills. The skill specialization reveals that agents in different roles undergo very unique experiences. Thus, sharing experiences between individuals in different roles, or letting individuals experience different roles, may be detrimental to their performance.

Nevertheless, the trends, if not the magnitude, of the changes, were similar across player. This suggests there is still benefit to social learning, or cross-agent communication of learning experiences. For instance, in the case of goalies, because they have fewer chances typically to intercept the ball, it is particularly useful to transfer mean values from the other players.

Finally, even though the on-line learning has not been active in competitions, the results of the learning were deployed in the final competitions of RoboCup'98. Our subjective evaluation is that it lead to a marked improvement.

6 Evaluating ISIS as an Agent Team in RoboCup

ISIS agents, as described in this article, are implemented in the hybrid architecture discussed in Section 2.2, where the higher-level is controlled by Soar, and the lower-level by C. Each ISIS agent currently contains about 665 Soar rules total, which include rules for all of the roles (goalie, forward, midfielder etc) as well as the general, domain-independent STEAM rules.

Previous sections describe our evaluation of ISIS's individual capabilities. This section focuses on the evaluation of the complete team that results. One aspect of this evaluation is the team's performance at RoboCup'97 (all of the capabilities described earlier, were present in our ISIS team at RoboCup'97, except for the learning of intercepts, which was developed after the competitions). As mentioned earlier, ISIS successfully participated in RoboCup'97, winning the third-place prize in the simulation league tournament, in over 30 teams that participated. Table 3 presents the results of ISIS from the first round of RoboCup'97. The first round had 8 groups (Group A to Group H). ISIS participated in group H. In this round, ISIS managed to easily defeat its opponents, in one-sided games.

Figure 7 presents the results of ISIS from the second round of RoboCup'97. As expected, ISIS met with stronger opponents in the second round. Here, ISIS won three games, and lost one game. Thus, over the seven games in the two rounds, ISIS lost just one game, but won six. It scored 37 goal, and had 19 goals scored against it.

A second aspect of evaluation of the ISIS team, given the goals of the RoboCup effort, is research contributions. To this end, this article has described the approach taken in ISIS to address the research challenges of RoboCup simulation league, in particular, the research challenges of multi-agent teamwork and learning. This article provides some evaluation of these approaches as well.

vs	ISIS	Team #2	Team #3	Wins/defeat	Ranking
ISIS	N/A	17-0	10-0	2/0	1
Team #2	0-17	N/A	4-0	1/1	2
Team #3	0-10	0-4	N/A	0/2	3
Team #4	-	-	-	-	-

Table 3: ISIS results in first round of RoboCup'97, simulation league. Team #2 is a private entry by Tomabechi. Team #3 is Innoue and Wilkin of Waseda University, Japan. Team #4 is Linkoping University, Sweden, which was a no-show at this RoboCup.

7 Related Work

There are two arenas of related work: (i) related work within RoboCup; and (ii) related work outside of RoboCup, in other multi-agent systems. Given the focus of this special issue on RoboCup, we will emphasize the related work in RoboCup, only briefly discussing the other related work.

With respect to teamwork in RoboCup, ISIS was the only team at RoboCup'97 that investigated the use of a general, domain-independent teamwork model to guide agent's communication and coordination in teamwork. Some researchers investigating teamwork in RoboCup have used explicit team plans and roles, but they have relied on *domain-dependent* communication and coordination. A typical example includes work by Ch'ng and Padgham[2]. They present an elaborate analysis of roles in motivating teamwork and team plans. In this scheme, agents dynamically adopt and abandon roles in pre-defined tactics. The responsibilities and actions of each agent are determined by its current role in the current plan. Unlike ISIS agents, whose team-related responsibilities are part of the general domain-independent STEAM model, Ch'ng and Padgham's roles include both team-level responsibilities as well as personal responsibilities — and so there is no separation of the domain-dependent from the domain-independent responsibilities. A similar scheme is used by Stone and Veloso [26]. They offer an approach to managing flexible formations and roles within those formations, allowing agents to switch roles and formations dynamically in a domain-dependent manner. Their agents synchronize their individual beliefs periodically in a fixed manner, in contrast with ISIS's STEAM in which communications are issued dynamically and can be parameterized based on the domain of deployment.

Another RoboCup effort focusing on explicit team plans and roles is [1]. They define levels of teamwork, starting at basic roles (which are static throughout the game), and building on top of those with formations and team plans for carrying out more complex tactics. These teamwork levels determine the agent's coordination responsibilities and prioritize its actions.

Other investigations of teamwork in RoboCup have used implicit or emergent coordination. A typical example is Yokota et al.[35]. They report on a hybrid technique for attaining physical robot cooperation: (i) negotiations-based cooperation utilizing explicit commu-

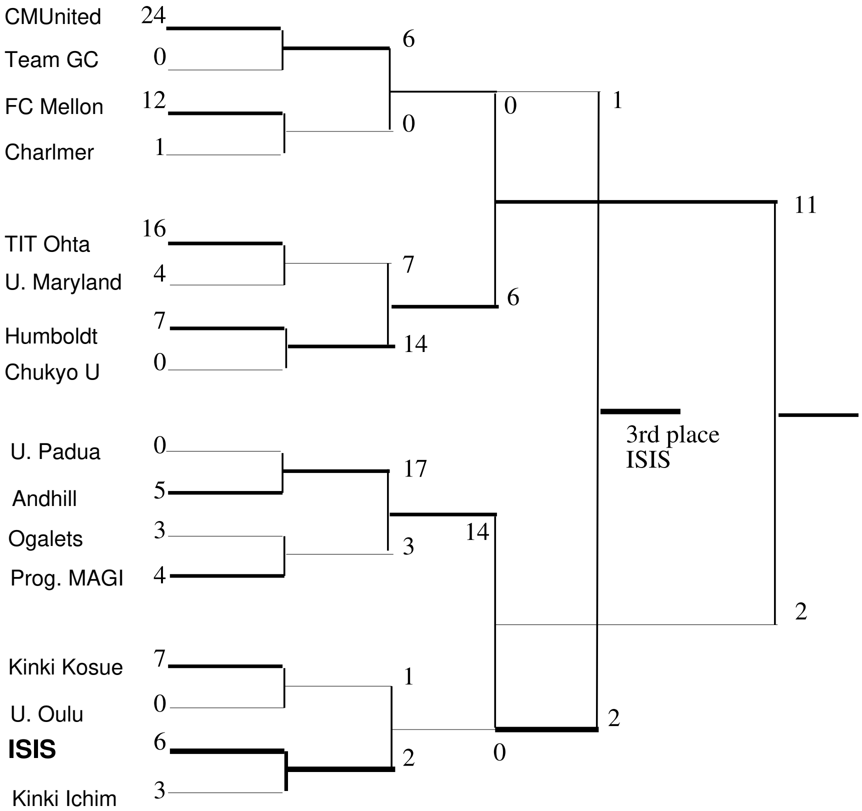

Figure 7: Results of the second round of RoboCup'97 simulation league.

nications, and (ii) sensing-based cooperation. Negotiations-based cooperation is achieved through explicit communications between the robots on the allocation of resources, conflict resolution, and exchange of information. The sensing-based cooperation is the "emergent" type coordination. It is based on the robots individually interpreting the sensed information and choosing actions such that their own self-interest is satisfied. They provide an example of how these two types of cooperation may work with a ball-pass between two robots.

Our application of learning in ISIS agents is similar to some of the other investigations of learning in RoboCup agents. For instance, Luke et al.[17] use genetic programming to build agents that learn to use their basic individual skills in coordination. Stone and Veloso[27] present a related approach, in which the agents learn a decision tree which enables them to select a recipient for a pass. The confidence values from the decision tree are also used to direct the agents actions – if no good recipient for a pass is suggested by the decision tree, the agent chooses a different, individual tactic.

In terms of related work outside RoboCup, the use of a teamwork model remains a distinguishing aspect of teamwork in ISIS. Indeed, teamwork models have only recently begun to emerge as a promising and useful approach to multi-agent collaboration. Thus, the STEAM teamwork model used in ISIS, is among just a very few implemented general models of teamwork. Other models include Jennings' *joint responsibility* framework in the GRATE* system[11] (based on Joint Intentions theory), and Rich and Sidner's COLLAGEN[23] (based on the SharedPlans theory), that both operate in complex domains. STEAM significantly differs from both these frameworks, via its focus on a different (and arguably wider) set of teamwork capabilities that arise in domains with teams of more than two-three agents, with more complex team organizational hierarchies, and with practical emphasis on communication costs (see [32] for a more detailed discussion). The other implementations of teamwork model emphasize different capabilities, e.g., COLLAGEN focuses on human-agent collaboration, and brings to bear capabilities more useful in such a collaboration.

8 Summary

This article focused on two important research issues in multi-agent systems: (i) teamwork and (ii) learning. With respect to teamwork, the article investigated a novel approach of the use of a domain-independent explicit model of teamwork, and the explicit representation of team goals and team plans. Teamwork models have recently been proposed as a promising approach to enhance teamwork flexibility and to enable reuse of teamwork capabilities[11, 32]. This article has empirically illustrated the usefulness of this teamwork-model-based approach in a novel domain. The teamwork model was shown to improve team performance, and reduce development and coding time. This teamwork-model-based approach is a general approach to teamwork, widely applicable in other domains beyond RoboCup. Indeed, the teamwork model applied in our work, STEAM, has also been applied for building teams of helicopter pilot agents for real-world combat simulations for training[32].

The article has also investigated the learning of low-level skills within a dynamic multi-agent setting. Both the off-line learning of goal kicking direction and on-line learning of intercept have been critical in improving ISIS agents. In some ways, our results also point out the importance of learning in "context". In the off-line learning, it was difficult for the expert to map their expertise of human soccer to a consistent labelling of instances which was appropriate for the context of a RoboCup player. Similarly, on-line learning revealed that different players, even the same player facing different opponents, learned different intercepts. Thus, not only does an outside expert have difficulty translating their expertise to what a player should do but also the skills of fellow players or even the same player may not translate directly to a particular context in a particular game. This appears to argue for a role-based learning approach based on an agent's unique experiences and for learning which is an ongoing process that adjusts to new experiences as opposed to converging on some fixed value.

Our team of soccer playing agents, ISIS, which was based on the above teamwork and learning capabilities, has successfully participated in the simulation league of the RoboCup'97 soccer tournament. We have taken a principled approach in developing ISIS, guided by the research opportunities in RoboCup. Despite the significant risk in following such a principled approach, ISIS won the third place in over 30 teams that participated in the RoboCup'97 simulation league tournament.

There are several key issues that remain open for future work. One key issue is improved agent- or team-modeling. One immediate application of such modeling is recognition that an individual, particularly a team member, is unable to fulfill its role in the team activity. In such cases, STEAM enables agents to reason about taking over others' roles, e.g., enabling a midfielder to take over the role of a non-performing forward. However, currently, in the absence of the required agent modeling capability STEAM cannot engage in such reasoning. Techniques for team modeling such as [30] would be applicable in this case. Indeed, this is partly the reason that many STEAM rules have currently not applied in RoboCup (so that STEAM reuse is limited to 45% of rules). A second key issue is improved agent and team learning, particularly to learn new team tactics. Further information about ISIS, including the code, is available at the following web site:

`www.isi.edu/teamcore/tambe/socteam.html.`

Acknowledgment

This research is supported in part by NSF grant IRI-9711665, and in part by a generous gift from the Intel Corporation. We thank Bill Swartout, Paul Rosenbloom and Yigal Arens of USC/ISI for their support of the RoboCup activities described in this paper. We also thank Peter Stone and Manuela Veloso for providing us player-agents of CMUnited, which provided a good opponent team to practice against in the weeks leading up to RoboCup'97.

References

[1] T. F. Bersano-Begey, P. G. Kenny, and E. H. Durfee. Agent teamwork, adaptive learning, and adversarial planning in robocup using a prs architecture. In *RoboCup-97: The first robot world cup soccer games and conferences*. Springer-Verlag, Heidelberg, Germany, 1998.

[2] S. Ch'ng and L. Padgham. Team description: Royal merlbourne knights. In *RoboCup-97: The first robot world cup soccer games and conferences*. Springer-Verlag, Heidelberg, Germany, 1998.

[3] P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 35, 1991.

[4] T. Dean, K. Basye, and J. Skewchuk. Reinforcement learning for planning and control. In *Machine Learning Methods for Planning*, pages 67–92. Morgan Kaufman, San Francisco, 1993.

[5] J. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1987.

[6] B. Grosz. Collaborating systems. *AI magazine*, 17(2), 1996.

[7] B. Grosz and S. Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–358, 1996.

[8] B. J. Grosz and C. L. Sidner. Plans for discourse. In P. R. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*, pages 417–445. MIT Press, Cambridge, MA, 1990.

[9] B. Hayes-Roth, L. Brownston, and R. V. Gen. Multiagent collaobration in directed improvisation. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*, 1995.

[10] F. F. Ingrand, M. P. Georgeff, , and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE EXPERT*, 7(6), 1992.

[11] N. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75, 1995.

[12] D. Kinny, M. Ljungberg, A. Rao, E. Sonenberg, G. Tidhard, and E. Werner. Planned team activity. In C. Castelfranchi and E. Werner, editors, *Artificial Social Systems, Lecture notes in AI 830*. Springer, NY, 1992.

[13] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of IJCAI-95 Workshop on Entertainment and AI/Alife*, 1995.

[14] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on autonomous agents*, 1997.

[15] H. Kitano, M. Tambe, P. Stone, S. Coradesci, H. Matsubara, M. Veloso, I. Noda, E. Osawa, and M. Asada. The robocup synthetic agents' challenge. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, August 1997.

[16] H. J. Levesque, P. R. Cohen, and J. Nunes. On acting together. In *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, Calif.: AAAI press, 1990.

[17] S. Luke, Hohn C., J. Farris, G. Jackson, and J. Hendler. Co-evolving soccer softbot team coordination with genetic programming. In *RoboCup-97: The first robot world cup soccer games and conferences*. Springer-Verlag, Heidelberg, Germany, 1998.

[18] S. Mahadevan and J. Connel. Automatic programming of behavior-based robots using reinforcement learning. In *Proceedings of the National Conference of the American Association for Artificial Intelligence (AAAI)*, August 1991.

[19] H. Matsubara, I. Noda, and K. Hiraki. Learning of cooperative actions in multi-agent systems: a case study of pass play in soccer. In S. Sen, editor, *AAAI Spring Symposium on Adaptation, Coevolution and Learning in multi-agent systems*, March 1996.

[20] A. Newell. *Unified Theories of Cognition*. Harvard Univ. Press, Cambridge, Mass., 1990.

[21] J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[22] A. S. Rao, A. Lucas, D. Morley, M. Selvestrel, and G. Murray. Agent-oriented architecture for air-combat simulation. Technical Report Technical Note 42, The Australian Artificial Intelligence Institute, 1993.

[23] C. Rich and C. Sidner. COLLAGEN: When agents collaborate with people. In *Proceedings of the International Conference on Autonomous Agents (Agents'97)*, 1997.

[24] P. S. Rosenbloom, J. E. Laird, A. Newell, , and R. McCarl. A preliminary analysis of the soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47(1-3):289–325, 1991.

[25] P. Stone and M. Veloso. Towards collaborative and adversarial learning: a case study in robotic soccer. In S. Sen, editor, *AAAI Spring Symposium on Adaptation, Coevolution and Learning in multi-agent systems*, March 1996.

[26] P. Stone and M. Veloso. Task decomposition and dynamic role assignment for real-time strategic teamwork. In *Proceedings of the international workshop on Agent theories, Architectures and Languages*, 1998.

[27] P. Stone and M. Veloso. Using decision tree confidence factors for multiagent control. In *RoboCup-97: The first robot world cup soccer games and conferences*. Springer-Verlag, Heidelberg, Germany, 1998.

[28] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[29] M. Tambe. Teamwork in real-world, dynamic environments. In *Proceedings of the International Conference on Multi-agent Systems (ICMAS)*, December 1996.

[30] M. Tambe. Tracking dynamic team activity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, August 1996.

[31] M. Tambe. Agent architectures for flexible, practical teamwork. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, August 1997.

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

[33] M. Tambe, W. L. Johnson, R. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), Spring 1995.

[34] M. Williamson, K. Sycara, and K. Decker. Executing decision-theoretic plans in multi-agent environments. In *Proceedings of the AAAI Fall Symposium on Plan Execution: Problems and Issues*, November 1996.

[35] K. Yokota, K. Ozako, Matsumoto A., T. Fujii, Asama H., and I. Endo. Cooperation towards team play. In *RoboCup-97: The first robot world cup soccer games and conferences*. Springer-Verlag, Heidelberg, Germany, 1998.