ISIS: Using an Explicit Model of Teamwork in RoboCup'97

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

Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way Suite 1001
Marina del Rey, CA 90292, USA
robocup-sim@isi.edu
www.isi.edu/soar/tambe/socteam.html

Abstract. Team ISIS (**ISI S**ynthetic) successfully participated 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 of RoboCup'97 (the most popular among the three RoboCup'97 leagues). In terms of research accomplishments, ISIS illustrated the usefulness of an explicit model of teamwork both in terms of reduced development time and improved teamwork flexibility. ISIS also took some initial steps towards learning of individual player skills. This paper discusses the design of ISIS in detail, with particular emphasis on its novel approach to teamwork.

1 Introduction

The ISIS (**ISI S**ynthetic) team of synthetic soccer-players won the third-place prize in the RoboCup'97 simulation league tournament. Developed at the University of Southern California's Information Sciences Institute (ISI), ISIS was also the top US simulation team. In terms of research accomplishments, ISIS illustrated the reuse of STEAM, a general model of teamwork[16], that both reduced its development time and improved teamwork flexibility.

ISIS's development is driven by the three research challenges emphasized in the RoboCup simulation league: (i) teamwork; (ii) multi-agent learning; and (iii) agent- and team-modeling[6]. With respect to teamwork, one key novelty in ISIS is its use of STEAM, a general, explicit model of teamwork to enable teamwork among player agents. This general model is motivated by the need for flexibility in team activities, as well as reuse of teamwork capabilities across domains[14, 15, 16]. STEAM uses the formal *joint intentions* framework[1, 7] as its basic building block, but with key enhancements to reflect the constraints of real-world domains. STEAM requires that individual team members explicitly represent their team's goals, plans and mutual beliefs. It then enables team members to autonomously reason about coordination and communication in teamwork, providing improved flexibility. Indeed, all of the current communication among ISIS agents is driven by STEAM's general purpose reasoning about teamwork. Given its domain-independence, STEAM also enables reuse across domains — here, RoboCup provided a challenging test domain, given its substantial dissimilarity from the original domain of STEAM's application (pilot teams for combat simulations for military training[16, 17]). Yet, a promising 35% of the original STEAM code was reused in RoboCup, and no new general-purpose teamwork code was required.

With respect to multi-agent learning, the second challenge in RoboCup, ISIS took some initial steps towards addressing it. Using C4.5[10], ISIS players learned off-line to choose an intelligent kicking direction, avoiding areas of concentration of opponent players. With respect to the third challenge, ISIS also performed limited agent- and team-modeling (particularly relevant to teamwork), but detailed plan-recognition of opponent-team's strategies remains an open issue for future work.

The rest of this paper is organized as follows: Section 2 describes the architecture of an individual ISIS agent. Section 3 describes the teamwork capability in ISIS. Section 4 discusses C4.5-based learning in ISIS. Section 5 then provides a summary and topics for future work. We will assume that the reader is familiar with Soccer, the RoboCup simulation league rules, as well as the RoboCup simulator[5].

2 ISIS Individual Agent Architecture

An ISIS agent is developed as a two-level architecture. The lower level, developed in C, communicates inputs received from the RoboCup simulator (after sufficient pre-processing), to the higher level. The lower level also rapidly computes some recommended directions for turning and kicking, to be sent to the higher-level. For instance, it computes three possible directions to kick the ball: (i) a group of C4.5 rules compute a direction to kick the ball towards the opponents' goal while avoiding areas of concentration of opponents; (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). The lower-level also computes a direction to turn if a player is to intercept an approaching ball.

The lower level 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 level, implemented in the Soar integrated AI architecture[9, 11]. Once the Soar-based higher-level reaches a decision, it communicates with the lower-level, which then sends the relevant information to the simulator.

The Soar architecture involves dynamic execution of an operator (reactive plan) hierarchy. These operators consist of (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/tasks and beliefs about its environment. Selecting high-level abstract operators for execution leads to subgoals, where new operators are selected for execution, and thus a hierarchical expansion of operators ensues. Activated operators 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[4], BB1[3], RAP[2] for dynamic domains may be similarly characterized in this fashion.

The operator hierarchy shown in Figure 1 illustrates a portion of the operator hierarchy for ISIS player-agents in RoboCup. One key novelty in this hierarchy, to support STEAM's teamwork reasoning (discussed below), is the inclusion of *team operators* (reactive team plans). Team operators explicitly express a team's joint activities, unlike the regular "individual operators" which express an agent's own activities. In the hierarchy in Figure 1, operators shown in boxes such as $\boxed{\text{WIN-GAME}}$ are team operators, while others are individual operators. The key here is that when executing team operators, agents bring to bear STEAM's teamwork reasoning, which facilitates their communication and coordination.

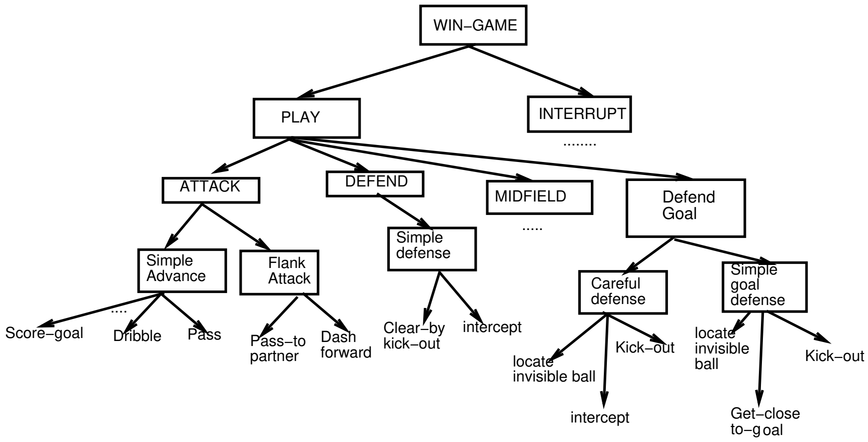

Fig. 1. A portion of the operator hierarchy for player-agents in RoboCup soccer simulation. Boxed operators are team operators, others are individual operators.

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. The team state is usually initialized with information about the team, such as the team members in the team, possible subteams, available communication channels for the team, the pre-determined team leader and so forth. STEAM can also maintain subteam states for subteam participation. 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, one key restriction is imposed for modifications to it — only the team operators executed by that (sub)team can modify it.

In Figure 1, the highest-level operator is a team operator called WIN-GAME. When all of the player agents select WIN-GAME in their operator hierarchy, the WIN-GAME team operator is established. There are two possible operators that can be executed in service of WIN-GAME, specifically PLAY (when the ball is in play) or INTERRUPT (when the ball is not in play). When the team operator PLAY is active, one of four team operators, ATTACK, MIDFIELD, DEFEND and DEFEND-GOAL can be executed. Each of these four is executed by a different subteam. Thus, the subteam of *forwards* in ISIS, typically consisting of three players, executes the ATTACK team operator. In service of ATTACK, the subteam may execute FLANK-ATTACK or SIMPLE-ADVANCE. In service of these, one of several individual operators, such as "score-goal" may be executed. Meanwhile, a second subteam of three agents may execute the DEFEND team operator. At any one time, an ISIS player agent has only one path through this hierarchy that is active, i.e., executed by the agent.

3 Teamwork in ISIS

As mentioned earlier, teamwork in ISIS is driven by a general, explicit model of teamwork called STEAM. STEAM uses the joint intentions theory as the basic building block of teamwork, and hence this theory is briefly discussed in Section 3.1. STEAM's communication and coordination activities, driven by this theory, are discussed in Section 3.2. STEAM was originally developed in the context of building teams of helicopter pilot-agents for real-world military simulations[16]. Originally developed within Soar, STEAM is currently encoded in the form of 283 Soar rules. RoboCup has provided a challenging domain for testing reuse of these STEAM rules, in a substantially dissimilar domain. Currently, about 35% of the rules are reused in ISIS, and this reuse may likely increase in the future.

3.1 Joint Intentions

STEAM's general model of teamwork is based on the *joint intentions* theory[7]. A joint intention of a team Θ is based on its joint commitment, which is defined as a joint persistent goal (JPG). A JPG to achieve a team action $\mathbf{p}$, denoted JPG(Θ, $\mathbf{p}$) requires all teammembers to mutually believe that $\mathbf{p}$ is currently false and want $\mathbf{p}$ to be eventually true.

JPG provides a basic change in plan expressiveness, since it focuses on a team task. Furthermore, a JPG guarantees that team members cannot decommit until $\mathbf{p}$ is mutually known to be achieved, unachievable or irrelevant. Basically, JPG(Θ, $\mathbf{p}$) requires team members to each hold $\mathbf{p}$ as a weak achievement goal

(WAG).[1] WAG(μ, **p**, Θ), where μ is a team member in Θ, requires μ to achieve **p** if it is false. However, if μ privately believes that **p** is either achieved, unachievable or irrelevant, JPG(Θ,**p**) is dissolved, but μ is left with a commitment to have this belief become Θ's mutual belief. Such a commitment helps to avoid communication failures — to establish mutual belief, an agent must typically communicate with its teammates.

Members of Θ must synchronize to establish JPG(Θ,**p**). To achieve such *team synchronization* we adapt the *request-confirm* protocol[12], described below. The key here is a persistent weak achievement goal (PWAG(νi, **p**, Θ)), which commits a team member νi to its team task **p** prior to a JPG. μ initiates the protocol while its teammates in Θ, ν1,.,νi..νn, respond:

1. μ executes a **Request**(μ, Θ, p), cast as an **Attempt**(μ, ϕ, ψ). That is, μ's ultimate goal ϕ is to both achieve **p**, and have all νi adopt PWAG(νi, **p**, Θ). However, μ is minimally committed to ψ, i.e., just to achieve mutual belief in Θ that μ has the PWAG to achieve ϕ. With this **Request**, μ adopts the PWAG.
2. Each νi responds via **confirm** or **refuse**. **Confirm**, also an **Attempt**, informs others that νi has the PWAG to achieve **p**.
3. If $\forall$ i, νi confirm, JPG(Θ, **p**) is formed. b

Besides synchronization, this protocol enforces important behavioral constraints. In step 1, the adoption of a PWAG implies that if after requesting, μ privately believes that **p** is achieved, unachievable or irrelevant, it must inform its teammates. Furthermore, if μ believes that the minimal commitment ψ is not achieved (e.g., the message did not get through) it must retransmit the message. Step 2 similarly constrains team members νi to inform others about **p**, and to rebroadcast. If everyone confirms, a JPG is established.

Thus, communication arises in the joint intentions theory to establish joint intentions, and to terminate them. However, communication in service of establishing and termination of each and every joint intention can be highly inefficient[16]. Hence, STEAM includes decision-theoretic communication selectivity. In particular, agents explicitly reason about the costs and benefits of communication, e.g., they avoid costly communication if there is a high likelihood that the relevant information can be obtained by other teammates via observation.

3.2 Joint Intentions in ISIS

Joint intentions are operationalized in STEAM via team operators. In particular, when all team members select a team operator such as WIN-GAME for execution (see Figure 1), they establish a joint intention. As participants in such a joint intention, STEAM enables individual team members to reason about their coordination and communication responsibilities.

Thus, based on the joint intentions theory, an individual cannot arbitrarily terminate a team operator on its own. Instead, a team operator can only be terminated if there is mutual belief that the operator is achieved, unachievable or irrelevant. Establishing such mutual belief in the termination of a team operator

[1] WAG was originally called WG in [7], but later termed WAG in [12].

can lead to communication. In particular, communication on termination of team operator arises if an agent privately realizes some fact relevant to the termination of a current team operator. Thus, if an agent's private state contains a belief that terminates a team operator (because it is achieved, unachievable or irrelevant), and such a belief is absent in its team state, then it creates a communicative goal, i.e., a communication operator. The generation of this communication operator is regulated based on its costs and benefits. When executed, this operator leads the agent to broadcast the information to the team.

Indeed, all communication in ISIS agents is currently driven by STEAM's general-purpose teamwork reasoning. A typical example of such communication is seen when three players in the "goalie" subteam execute the DEFEND-GOAL team operator. In service of DEFEND-GOAL, players in this subteam normally execute the 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 repositioning itself, can be unaware of the approaching ball. Here is where teamwork can be beneficial. In particular, if any one of these players sees the ball as being close, it declares the SIMPLE-DEFENSE team operator to be irrelevant. Its teammates now focus on defending the goal in a coordinated manner via the 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 CAREFUL-DEFENSE is irrelevant). The subteam players once again execute SIMPLE-DEFENSE to attempt to position themselves close to the goal. In this way, agents attempt to coordinate their defense of the goal, while also attempting to position themselves near it.

4 Learning

Inspired by previous work on machine learning in RoboCup[13, 8], we focused on techniques to improve individual players' skills to kick, pass, or intercept the ball. Fortunately, the two layer ISIS architecture helps to simplify the problem for skill learning. In particular, the lower-level in ISIS is designed to provide several recommendations (such as several alternative kicking directions) to the higher-level, but it need not arrive at a specific decision (one specific kicking direction). Thus, an individual skill, such as a kicking direction to clear the ball, can be learned independent other possible actions. That is, the learning algorithm is not forced to simultaneously learn to select if clearing the ball is the best choice among available alternatives. Instead, that decision is left to the higher-level.

For the RoboCup'97 tournament, C4.5[10] was successfully used in ISIS to learn to select an *intelligent* kicking direction. C4.5 rules were learned off-line via a batch of training examples to select a direction to kick towards the opponent's goal while avoiding areas of concentration of opponent players. This learned kicking direction was one among three kicking directions computed in the lower-level (as discussed in Section 2). The higher-level typically selected the learned kicking direction (from the three provided to it) when players were close to the goal, and were ready to directly score a goal. While we did initial exploration to learn the remaining two kicking directions, those results were not ready for RoboCup'97. We hope to field a team with further learned skills for RoboCup'98.

5 Summary

The overall goal in ISIS was not and has not just been one of building a team that wins the RoboCup tournaments. Rather, ISIS has taken a principled approach, 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. For instance, if a forward is "covered" by the opponents, it may be unable to fulfill its role. 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. Indeed, this is partly the reason that many STEAM rules have currently not applied in RoboCup (so that STEAM reuse is limited to 35% of rules).

A second key issue arose as a lesson learned from ISIS's participation in RoboCup'97. A weakness was discovered in ISIS, that stemmed from a somewhat inappropriate interaction with the RoboCup simulator — the simulator version used in RoboCup'97 allowed agents to take up to three actions (one action per 100 ms) before sending them a sensor update (one update per 300 ms). This required that agents continually make predictions. Unfortunately, with weak predictive capabilities, ISIS agents could not always quickly locate and intercept the ball, or maintain awareness of positions of teammates and opponents. This key weakness was a factor in the single loss that ISIS suffered in the course of the RoboCup'97 tournament. However, the RoboCup simulator will evolve for RoboCup'98, towards more human-like play.

Overall, we hope to continue working on ISIS in preparation for RoboCup'98, and meet the research challenges outlined for the simulation league in teamwork, multi-agent learning and agent modeling[6]. Further information about ISIS, including the code, is available at the following web site:

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

STEAM code, with detailed documentation and traces is available at:

`www.isi.edu/soar/tambe/steam/steam.html`

ISIS team members can be reached at robocup-sim@isi.edu.

Acknowledgement

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. P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 35, 1991.
2. J. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1987.
3. 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.
4. F. F. Ingrand, M. P. Georgeff, , and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE EXPERT*, 7(6), 1992.
5. 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.
6. 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.
7. 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.
8. 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.
9. A. Newell. *Unified Theories of Cognition*. Harvard Univ. Press, Cambridge, Mass., 1990.
10. J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, CA, 1993.
11. 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.
12. I. Smith and P. Cohen. Towards semantics for an agent communication language based on speech acts. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, August 1996.
13. 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.
14. M. Tambe. Teamwork in real-world, dynamic environments. In *Proceedings of the International Conference on Multi-agent Systems (ICMAS)*, December 1996.
15. M. Tambe. Agent architectures for flexible, practical teamwork. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, August 1997.

16. M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR)*, 7:83–124, 1997.
17. 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.