A framework for exploring role assignment in real-time, multiagent teams

Vanessa Frias-Martinez and Elizabeth Sklar
Department of Computer Science
Columbia University
1214 Amsterdam Avenue, Mailcode 0401
New York, NY 10027, USA
vf2001, sklar@cs.columbia.edu

Abstract

We are examining the general problem of resource allocation, in particular, role assignment and task organization within real-time, multiagent teams. Our current work involves developing a framework for studying this problem, focusing on methodologies for administration of tasks and coordination of team members operating in highly dynamic environments where changes occur frequently and rapidly. This paper outlines initial work in this area, describing the architecture of our "plug-and-play" framework and specific freeware components that we have combined to create a flexible environment for experimentation.

1. Introduction

We are examining the general problem of resource allocation, in particular, role assignment and task organization within real-time, multiagent teams. Our current work involves developing a framework for studying this problem, focusing on methodologies for administration of tasks and coordination of team members operating in highly dynamic environments where changes occur frequently and rapidly. One of our application domains is the RoboCup Four-Legged Soccer League [14]. Eventually, we will deploy our strategies on the physical robots: Sony AIBO ERS-210's and ERS-7's (see figure 1); but for now, we are experimenting in simulation. In developing our simulation environment, we have encountered a wide body of literature and a varied set of tools for simulating various aspects of multiagent systems (MAS). This has motivated us to design a "plug-and-play" framework that allows one to explore any number of these components in depth, and at the same time, apply them to different domains.

(a) Model ERS-210

(b) Model ERS-7

Figure 1. Sony AIBO robots.

There are four basic components to our framework, as shown in figure 2:

- application environment
- resource allocation mechanism
- learning methodology
- multiagent simulator

The *application environment* represents the real-time, dynamic domain in which the multiagent team is acting, such as robotic soccer or other domains with similar characteristics, for example, grid node management in operating systems. The *resource allocation mechanism* is where decisions are made about task organization and role assignment. Here is where we can ex-

periment with different techniques for reaching decisions, such as auctions or argumentation. The *learning methodology* indicates which class of representations and machine learning algorithms are being used to enable the team to learn how to behave better, such as genetic algorithms or neural networks, and reinforcement or co-evolutionary learning. The *multiagent simulator* controls the interaction between the other components and simulates time passing, giving the agents turns to sense, plan, act and learn in the application environment.

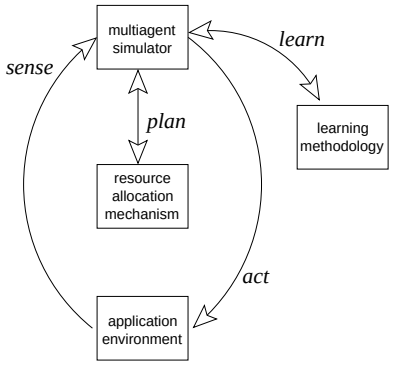

Figure 2. Our framework.

Our framework allows us to take advantage of existing tools and/or techniques for each component and "plug them in", adapting them to the specifics of our problem domain — here, we focus on robot soccer. To implement the application environment, we have been using the robotic simulator *PlayerStage* [3]. For making decisions about role assignment, we are using an auction-based mechanism in which agents continually bid for roles as their environment changes. To effect clearing of the bids, we are using the *JASA* package [17]. Finding successful bidding strategies is quite difficult, and so we are using genetic algorithms to learn how to bid effectively [7]. Linking all these pieces together is the multiagent simulator, implemented using the *RePast* package [16].

This paper is organized according to our framework, detailing the specific components we are using to develop team coordination in the RoboCup Four-Legged Soccer League. We begin by describing the application environment, then discuss the resource allocation mechanism that we are developing. Next we outline the methodology we have been using for enabling the agents to learn. Finally we describe the multiagent simulator that we have incorporated into the framework. We close with a brief summary and plans for future work.

2. Application environment

Techniques from the field of multiagent systems have been applied to robotics with increasing frequency over the last ten years [22]. Soccer-playing robot teams such as those participating in the RoboCup initiative [11] are a good training ground for a multiagent system: a set of heterogeneous agents, playing different positions on the soccer field, collaborate in a team and at the same time compete against an opposing team. A very important issue in the operation of a heterogeneous multiagent team is the assignment of roles to team members. In a highly dynamic environment, it is necessary to define a mechanism so that agents can change roles as the environment changes. Many RoboCup teams use hand-coded solutions to this problem.

We are exploring the use of an automated auction mechanism in which the bidding strategy is learned by the players using a co-evolutionary algorithm. In order to make progress with any machine learning algorithm, it is necessary to perform many runs over time. Typically, this is impractical and/or infeasible to do on real robots, due to physical constraints on the robots themselves and issues such as battery life; although we note that some quite clever work has been done to try and overcome these issues [23, 13, 5]. Nonetheless, having a simulator allows us to run experiments rapidly and obtain fairly accurate predictions in terms of physical measurements.

We have implemented the robot soccer application environment using a robotic simulation package called *PlayerStage* [3]. PlayerStage is an open source tool composed of two components, *Player* and *Stage*. *Player* acts as a *client* and provides a simulator base and network interface for a variety of robot and sensor hardware. There are existing Player clients that simulate different types of popular robots, including Pioneer and ATRV Jr. *Stage* acts as a *server* and simulates a population of mobile robots moving and sensing a two-dimensional environment. Stage provides a set of simulated sensors including sonar, scanning laser rangefinder, pan-tilt-zoom camera with color blob detection and odometry. Any number of Players can connect to the Stage, receive sensory information, perform planning, learning, communication, etc.; and then send action commands to the Stage to act out.

One of the great advantages of PlayerStage is the modularity of its design. Because the Stage and Player components run independently and communicate via network sockets, it is possible to run them on different platforms and to build Player clients in different languages. It is even possible to run Player clients on-

board real, physical robots, thus facilitating a unique mix of simulated and true, embodied agents.

This last feature is one of the primary reasons we have introduced PlayerStage into our framework. It will greatly simplify our future plan to move our simulated results onto real robots. In order to do this, we can implement a Player client running directly on an AIBO, which can talk to the Stage in real-time. Currently, there is not a Player client that simulates an AIBO, although we will be exploring techniques for automatic programming of such after we build the Player client that executes directly on the AIBO and collect runtime data.

3. Resource allocation mechanism

Many authors have studied automatic bidding strategies for agent-based systems applied to areas other than robot soccer, for example, fishmarket auctions [19] and trading agent competitions (TAC) [18] with adaptable single-agent auctions or competitive negotiation scenarios [25]. Those applications were created to run in software only. We believe that the first proven applications of auctions in physical multi robot systems were MURDOCH, developed by Gerkey and Matarić [9] and Traderbots, developed by Dias and Stentz [6].

In our robot soccer environment, we are experimenting with learning bidding strategies and auction-clearing mechanisms for role assignment. During the simulated soccer game, players place bids for roles, based on their perceptions of the current state of the field: e.g., have I seen an opponent? where is the ball? is the goal too far away? The auctioneer takes each agent's bid for which role to play at that moment in the game, coordinates the bids, factors in the state of the game, and returns roles to each agent.

JASA [17] is an open source tool for performing all sorts of simulations of auctions. In our setup, JASA performs the auction-clearing strategy for our "auctioneer", assigning roles based on players' bids and resolving conflicts which may arise. For example, two agents may want to be assigned the same role at the same time, which may be illegal or undesirable, depending on the constraints we define for the system.

Note that in our setup, the auctioneer is also part of the team (it is actually executed on the goalie robot), so its goals are the same as that of the players, which may somewhat narrow the practical strategy search space. In a typical auction, agents bid *against* each other for selfish rewards and are not concerned with the overall outcome of all agents participating in the auction. Here, we have an orthogonal situation: although agents place individual bids, their overall goal is shared by the other participants in the auction and so agents' bids will likely be more compatible than would otherwise be found in other types of auctions where some participants win at the expense of others. We refer to this as *team-based learning* and discuss this aspect in section 4.

The specifics of the bidding mechanism for our soccer robots is as follows. We have defined three different roles that agents can bid for: *primary attacker* (PA), *defensive supporter* (DS) and *offensive supporter* (OS). There are four players on each team. Note that the role of *goalie* is always fixed to one particular agent[1]; and in our case, as above, the goalie is also the auctioneer. It is not a requirement that there only be one agent assigned to each of the other three roles. Indeed, part of our experimentation involves considering questions like when is it better to have two, or even three, primary attackers on the field.

The (non-goalie) players construct a bidding strategy according to their perceptions of the state of the soccer field. For the purpose of conducting preliminary experiments, we greatly simplified the perception state of each agent to three binary values: (1) can I see the ball? (2) am I the closest player to the ball? (3) am I the closest player to our opponent's goal? These relative perceptions are represented by a 3-digit *percept code*, containing one bit per perception, which is set to 0 or 1 depending on the state of the field. Table 1 illustrates the eight possible relative perceptions and corresponding percept code.

ball seen?	closest to ball?	closest to goal?	percept code
no	no	no	000
no	no	yes	001
no	yes	no	010
no	yes	yes	011
yes	no	no	100
yes	no	yes	101
yes	yes	no	110
yes	yes	yes	111

Table 1. Percept code definitions.

An agent's *bidding strategy* defines a *bid code* that corresponds to each possible percept code value, and a bid consists of a preference ordering of (one, two or all three) roles. In our preliminary experiments, we used

1 The RoboCup soccer rules specify that one goalie per team must be designated *a priori*.

combinatorial auctions (each agents' bid contains a list of three roles) — offering second and third choices of role if the preferred role(s) were lost. Given three roles, six orderings are possible for combinatorial auctions of the three roles (without repetition). Table 2 contains the possible preference orders and the corresponding *bid code*, given the limitation that an agent's bid cannot contain duplicate roles. In future work, we will experiment with lifting this restriction.

role ordering	bid code
PA-OS-DS	0 (000)
PA-DS-OS	1 (001)
OS-PA-DS	2 (010)
OS-DS-PA	3 (011)
DS-PA-OS	4 (100)
DS-OS-PA	5 (101)

Table 2. Bid code definitions.

With this setup, the search space of all possible bid strategies for one agent (for the 8 perceptions and the 6 different role bids) is[2]:

$$VR(6,8) = 6^8 = 1.6 * 10^6, \text{ possible bids} \quad (1)$$

These calculations are for only one agent. Each team is composed of three role-choosing agents. This means that each of the possible bids that an agent can make are going to be combined with two others to make a team bid. In these terms, we have more than a million possibilities to be considered by a team of three:

$$VR(1.6 * 10^6, 3) > 10^{18}, \text{ possible bid teams} \quad (2)$$

Given this combinatorial explosion within the bidding space — even within our highly simplified experimental setup — it is a clear case where a machine learning algorithm can be helpful for identifying strong bidding strategies.

As described below, the agent bidding strategy is set at the beginning of a generation in the machine learning process; and a series of games is played to comprise a generation. During game play, the agents place their bids, at which time JASA is invoked to evaluate the bids and "clear" the auction. Thus far, we have experimented with a combinatorial auction where agents bid on a preference-ordering for roles. JASA allows us to experiment easily with other auction mechanisms as well.

2 where $VR(n,p) = n^p$ is the formula for computing the number of variations with repetition of selecting an ordered set of p elements from a set of possible n elements.

4. Learning methodology

The field of *multiagent learning* is described in [24] and in [21] as a fusion between multiagent systems and machine learning (ML). Applying ML techniques to MAS allows us to build evolving agents: agents that learn from their experiences and interactions, and adapt to their environment. Multiagent learning techniques were originally applied to one agent at a time [2], even as part of a multiagent system. Some research into learning as a team within an MAS was done by Nagendra et al. [15] and was one of the early attempts at demonstrating the utility of self-organization in a multiagent system where the agents are driven to achieve a common objective. Sen and Sekaran [20] applied reinforcement learning techniques to a multiagent box-pushing system. Our approach differs because we define our agents to be inherently selfish, but they must learn to act as contributing members of a team; and it is up to the auctioneer to balance the selfish requests of the individuals with the needs of the team. We use *genetic algorithms* (GA) [12] and a *co-evolutionary learning* [10, 1] to learn the combination of strategies that works best for a team.

All four agents learn their bidding and clearing strategies over time by playing many games and evaluating the results. The term *co-evolution* refers to evolving agents where the *fitness* of an individual (or a team) is based on that individual's (or team's) performance in their environment compared to that of another individual (or team) operating in the same environment; whereas standard *evolutionary learning* is based on a pre-defined, fixed performance measure. In instances of successful co-evolutionary learning, a virtual arms race occurs where agents of subsequent generations challenge each other, resulting in continuous improvement over time [4]. We consider the performance in the environment to mean the result of one team playing a series of soccer games against another team, and the fitness of the team is shared as the fitness for each individual agent on the team; we refer to this method of fitness sharing as *team-based learning*.

We implement the GA as follows; further detail can be found in [8]. Recall from the previous section that an agent's bidding strategy consists of a bid code for each of the eight percept codes. There are six possible bid codes, which can be represented by a 3-bit value 000 (0) and 101 (5). Assigning one of these codes for each of the eight precept codes can be represented by an $8x3 = 24$-bit string. The GA begins by randomly initializing bidding strategies for a population of n players (where $n \geq 6$); i.e., randomly setting bits in n 24-bit strings. At each generation, we randomly

select six players (3 players per team) from this population. Then the two teams play a series of games against each other, called a "round". Each round consists of g games; each generation consists of r rounds. The games are played for a limited amount of simulated time, and after each game, the fitness of the three agents belonging to the winning team is increased. After r rounds, a new generation is obtained by selecting the m best agents in the population ($m < n$), i.e., those with the highest fitness values, and using standard reproductive operators on these m agents to produce a new population of size n.

5. Multiagent simulator

The glue that holds the framework together and coordinates between the other three components is the multiagent simulator, in this case, *RePast*. We chose RePast because of its flexible architecture and rich set of tools for representing agents, collecting data and displaying results while a simulation is running. RePast is written in Java, which meant that we could fairly easily interface it with the other open-source components which are also Java based, namely Stage and JASA.

Experiments are conducted as follows. The genetic algorithm, implemented in Java as part of the RePast controller, begins by randomly defining n 24-bit strings, representing the initial population of players. The simulator selects two teams from the population and plays a round of games between them to make a generation. Game play is simulated using PlayerStage. RePast sends a message to PlayerStage saying it is ready for the game to begin. PlayerStage simulates a game, using the JASA package and shared variables to implement clearing of the auction each time a new set of perceptions comes in and role assignment takes place. When the game is over, PlayerStage sends the results back to RePast, which updates its statistics and then selects another pair of teams to play the next round of games.

6. Summary

We have described a framework that we have developed for conducting multiagent team-based learning experiments with role assignment in a highly dynamic environment. Our methodology provides a flexible, "plug-and-play" schema so that different mechanisms for role assignment, learning and even application domain can be tested. The setup takes advantage of existing tools and techniques; and here we have described our use of RePast, PlayerStage and JASA to experiment with learning roles within a robotic soccer team. In earlier work [8], we ran experiments only in RePast; however we believe this new, modular and flexible framework will provide us with opportunities to produce more applicable and robust results by allowing us to develop and test algorithms with a wider range of techniques and apply them to more domains.

This work is obviously preliminary, and our plan for tomorrow is to use the framework for conducting not only a broader series of experiments than those we have run so far using the precise setup outlined herein, but also introducing embodied agents into the learning process, crossing over to other application environments (such as grid node management) and evaluating different techniques for resource allocation.

7. Acknowledgments

This work was made possible by funding from NSF #IIS-03-29037.

References

[1] P. J. Angeline and J. B. Pollack. Competitive environments evolve better solutions for complex tasks. In S. Forrest, editor, *Genetic Algorithms: Proceedings of the Fifth International Conference (GA93)*, 1993.

[2] T. Balch. Learning roles: Behavioral diversity in robot teams. In *AAAI Workshop on MultiAgent Systems*, 1997.

[3] B.Gerkey, R. Vaughan, and A. Howard. *PlayerStage*. http://playerstage.sourceforge.net.

[4] A. Blair, E. Sklar, and P. Funes. Co-evolution, determinism and robustness. In S. Verlag, editor, *Proceedings of the SEAL-98 (Simulated Evolution and Learning Conference), Lecture Notes in Artificial Intelligence 1585*, 1998.

[5] S. Chernova and M. Veloso. An Evolutionary Approach To Gait Learning For Four-Legged Robots. In *Proceedings of IROS'04*, 2004.

[6] M. Dias and A. Stentz. A free market architecture for distributed control of a multirobot system. In *6th International Conference on Intelligent Autonomous Systems (IAS-6)*, 2000.

[7] V. Frias-Martinez and E. Sklar. A team-based coevolutionary approach to multi agent learning. In *In Workshop Proceedings of the Third International Conference on Autonomous Agents and Multi Agent Systems*, 2004.

[8] V. Frias-Martinez, E. Sklar, and S. Parsons. Exploring auction mechanisms for role assignment in teams of autonomous robots. In *Proceedings of the Eighth International RoboCup Symposium*, 2004.

[9] B. Gerkey and M. Mataric. Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5), 2002.

[10] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In L. et al., editor, *Proceedings of ALIFE-2*, pages 313–324. Addison Wesley, 1992.

[11] H.Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. The robot world cup initiative. In *In Proceedings of Autonomous Agents*, 1997.

[12] J. H. Holland. *Adaption in Natural and Artificial Systems.* University of Michigan Press, 1975.

[13] G. S. Hornby, S. Takamura, J. Yokono, O. Hanagata, T. Yamamoto, and M. Fujita. Evolving Robust Gaits with AIBO. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3040–3045, 2000.

[14] H. Kitano, Y. Kuniyoshi, I. Noda, M. Asada, H. Matsubara, and E. Osawa. RoboCup: A challenge problem for AI. *AI Magazine*, 18:73–85, 1997.

[15] M.V.Nagendra, V.Lesser, and S.Lander. Learning organizational roles in a heterogeneous multi-agent system. *Proceedings of the Second International Conference on Multi-Agent Systems*, 1997.

[16] U. of Chicago. *RePast.* http://repast.sourceforge.net.

[17] S. Phelps. *JASA.* http://jasa.sourceforge.net.

[18] S. S. P.Stone, M. Littman and M. Kearns. Attac-2000: An adaptive autonomous bidding agent. *Journal of Artificial Intelligence Research*, pages 189–206, 2001.

[19] J. Rodriguez-Aguilar, F. Martin, P. Noriega, P. Garcia, and C. Sierra. Towards a test-bed for trading agents in electronic auction markets. *AI Communications*, 1997.

[20] S. Sen and M. Sekaran. Learning to coordinate without sharing information. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 426–431, 1994.

[21] P. Stone and M. Veloso. Towards collaborative and adversarial learning: A case study in robotic soccer. *International Journal of Human Computer Studies*, 48:83–104, 1998.

[22] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robotics*, 8(3), July 2000.

[23] R. A. Watson, S. G. Ficici, and J. B. Pollack. Embodied Evolution: Embodying an Evolutionary Algorithm in a Population of Robots. In *1999 Congress on Evolutionary Computation*, pages 335–342. IEEE Press, 1999.

[24] G. Weib. Distributed reinforcement learning. *Robotics and Autonomous Systems*, 15:135–142, 1997.

[25] D. Zeng and K. Sycara. Bayesian learning in negotiation. In adaptation, coevolution and learning in multiagent systems: Papers from the 1996 aaai spring symposium, AAAI Tecnical Report SS-96-01, 1996.