Carnegie Mellon University
Research Showcase @ CMU

Robotics Institute | School of Computer Science

2002

Principled Monitoring of Distributed Agents for Detection of Coordination Failure

Brett Browning
Carnegie Mellon University

Gal A. Kaminka
Carnegie Mellon University

Manuela M. Veloso
Carnegie Mellon University

Follow this and additional works at: http://repository.cmu.edu/robotics

 Part of the Robotics Commons

This Conference Proceeding is brought to you for free and open access by the School of Computer Science at Research Showcase @ CMU. It has been accepted for inclusion in Robotics Institute by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

Principled Monitoring of Distributed Agents for Detection of Coordination Failure

Brett Browning, Gal A. Kaminka, and Manuela M. Veloso

School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213
{brettb, galk, veloso}@cs.cmu.edu

Abstract. There is a very rich variety of systems of autonomous agents, be it software or robotic agents. In particular, multi-agent systems can include agents that may be part of a team and need to coordinate their actions during their distributed task execution. This coordination requires an agent to observe, i.e., to monitor, the other agents in order to detect a possible coordination failure of the team. Several researchers have addressed the problem of monitoring for single or multiple agent systems and have contributed successful, but mainly application-specific, approaches. In this paper, we aim at contributing a unifying, domain-independent statement of the distributed multi-agent monitoring problem. We define the problem in terms of a pre-defined *desirable joint state* and an *observation-state mapping*. Given a concrete joint observation during execution, we show how an agent can detect a possible coordination failure by processing the observation-state mapping and the desirable joint state. To illustrate the generality of our formalism, one of the main contributions of the paper, we represent several previously studied examples within our formalism. We note that basic failure detection algorithms can be computationally expensive. We further contribute an efficient method for failure detection that builds upon an off-line compilation of the principled relations introduced. We show empirical results that demonstrate this effectiveness.

1 Introduction

Agents in distributed systems may need to coordinate. In the absence of allowed communication between the agents, coordination needs to be based on observations of the other agents. An agent therefore needs to monitor the other agents. Observation-based coordination (OBC) is a key challenge to the multi-agent and multi-robot systems. Increasingly, robots and synthetic agents are being deployed in multi-agent virtual environments for training [1] and entertainment [2], robotic soccer [3], hazardous cleanup tasks [4], formation-maintenance tasks [5,6], and more. Many of these applications rely on agents to coordinate with one another based on their observations of each other [7].

OBC is often a challenging process, mainly because it is computationally expensive. In general, OBC requires an agent to observe its peers and infer their state. Typically, the actual state is hidden and it is only partially revealed through observations. The agent must then decide what action it should take based on its own goals and internal state, and the observation-based inferred state of its peers. Specific examples of this process include OBC in self-interested agents [2], observation-based teamwork [8]. The inference and decision process are often considered computationally

too intense for resource-limited robots. Indeed, there have been many investigations of ways to generate robust and predictable globally-coordinated behavior using agent-local control rules that shortcut the inference and decision processes, (e.g. [6,5]). However, these approaches often require painstakingly hand-crafted control rules and have been applied mostly in spatial coordination tasks (see Section 2 for an in-depth exploration of OBC in the literature).

We focus in this paper on an important component of OBC: *observation based monitoring*, which is used when agents use observations to verify that a group of observed agents does not suffer from a coordination failure. We thus limit ourselves to determining the existence of a coordination failure, and do not consider the actions that the agent may take to respond to it (if detected).

We introduce a novel formalism for describing observation-based monitoring of *simultaneous activities*. This formalism can be used to describe a significant class of coordination processes reported in the literature and commonly found in real-world multi-agent systems. We use this formalism to describe on-line coordination, and explore the complexity of this task. We then show how an off-line compilation process emerges as a result from this description, essentially transferring the on-line run-time complexity of the task off-line, allowing for quick execution. To evaluate our work empirically, we examine several examples of OBC in the literature. We present simulation results demonstrating the significant run-time computational resources saved by the off-line computation process. This compilation process takes a first step towards unifying the two themes of observation-based coordination.

This paper is organized as follows: Section 2 presents an overview of previous investigations into the OBC process. Section 3 presents a formalism describing OBC. Section 4 how OBC can be compiled into reactive rules. Section 5 presents an evaluation, and Section 6 concludes.

2 Observation-Based Coordination

Observation-based coordination (OBC) begins by each agent observing its fellow agents. Observations can be as simple as relative distance, or location, or as complex as intended goal, or plan. Based on these observations, each agent decides on what actions it will perform so as to achieve its own goals. Note that we are using the phrase coordination here in its broadest sense: We consider agents to be coordinating when the actions of each agent are dependent on the state of the other agents. Thus, both collaborative and adversarial settings are included in this definition.

Work on OBC has traditionally followed two parallel themes. The first such approach, which we call *state-based coordination* (SBC), emphasizes coordination at the level of internal states of agents, e.g., at the level of their selected plans, goals, or behaviors. Thus, SBC requires that the (unobservable) internal state of observed agents be known to the coordinating agent. The other approach, which we call *reactive coordination* (RC), shortcuts this process by directly mapping from observations of others to subsets of actions to be executed. It is distinct from SBC in that it does not explicitly consider the internal state of the observed agents.

SBC uses explicit knowledge of the internal state of the observed agents. Unfortunately, this knowledge is typically inaccessible. The agent must therefore use whatever observations it has of the other agents to *infer* the state of the other agents from the observations. Based on this inferred state, the coordinating agent selects its own internal state and actions. A key benefit of state-based coordination is that the designer can specify such selections in a *coordinating policy*. A coordinating policy is often easier to understand and design since it relates the plans and goals of others to those of the coordinating agent at a convenient level of abstraction.

Unfortunately, SBC often requires substantial computational resources. Inferring the state of the other agents based on observations (a process sometimes known as plan-recognition [2], behavior recognition, or agent modeling) is often expensive because, in general, the same observation can be interpreted as evidence for multiple internal states. A common approach relies on probabilistic networks for plan recognition, e.g., [2]. Washington[9] relies on POMDPs in coordination. Both probabilistic networks and POMDPs are intractable in the general case. Washington shows that under certain conditions, the coordination problem using POMDPs can be polynomial, however these conditions require the coordinating agent to not affect the behavior of the observed agents–a significant restriction in collaborative and adversarial settings. Tambe developed the non-probabilistic polynomial-time $RESC_{team}$ algorithm for reasoning about adversaries hierarchical behaviors [10]. This algorithm gains its computational advantage by always adapting and committing to a single interpretation of the opponents' actions.

Independently from investigations of SBC (mostly in software agent settings), researchers in the multi-robot community have focused their efforts on approaches appropriate for resource-limited hardware. The key ideas in these is to shortcut the inference and decision making process of SBC by introducing *reactive coordination* (RC) behaviors that tie specific observations of other agents with actions by the coordinating agent. For instance, Mataric demonstrated that many spatial group behaviors can be achieved by combinations of relatively simple agent-local rules, that directly tie spatial observations (e.g., distance and angle to other robot) with actions that should be taken [6]. Balch investigated methods for reliable execution of group tasks, such as foraging and formation maintenance, using hierarchical reactive behaviors that emphasized coordination by reliance on simple relative observations of other agents [5]. Unfortunately, while RC does indeed offer significant execution-time advantages for resource-constrained settings, it is difficult to design and maintain. Coordinated group behavior emerges in RC out of the interaction of reactive behaviors, but these interactions are difficult to predict.

This paper takes a step towards unifying these themes, by showing how an important class of coordination relationships can be specified in a more understandable manner (i.e., at the internal state level), but still executed quickly by resource-constrained agents. We focus on an important component in the OBC task: The observation-based monitoring that allows agents to detect coordination failures in their peers, thus allowing them to decide on an action if a failure is detected. The idea is to compile the expensive on-line failure detection process at design time

into a set of fast-to-execute reactive rules. These rules can then be used at run-time to detect failures directly from observations without having to infer the joint agent state.

3 A Formalism for Describing Observation-Based Coordination

This section defines OBC formally and uses a running example from a coordination process from the literature to illustrate its meaning.

3.1 Examples from the literature

These examples are taken from the ModSAF domain, a high-fidelity virtual environment for military training that allows thousands of agents (synthetic and human) to interact in battlefield scenarios [1]. Several coordination scenarios for synthetic helicopter pilots in this domain have been described in [11,8]. These rise from the scenario described below.

> A team of 3–6 helicopters take off from their base and fly in formation (with the *fly-flight-plan* (f) behavior) until they reach an area marked by a visible landmark. Upon arrival, they select the *wait-at-point* (w) behavior, in which they split into two subteams. The scouting subteam moves towards the estimated enemy location to identify its exact position (l). The attacker subteam stays behind (b). When the scouting team identifies the enemy position, it radios for the attackers to join it (all team-members select the *join-scout* (c) behavior) and waits for it to arrive. Once joined, they engage the enemy together (the *engage* (e) behavior). Each helicopter *masks* (m) by hiding behind trees or hills and *unmasks* (u) to come out of hiding to *shoot* (s) at the enemy. It then masks again before moving to a new location to shoot. When the enemy is destroyed, the entire team joins together and flies in formation (f) back to base. At any point during the mission, pilots may be ordered to *halt* (h) their activities and await further instructions.

The designers of these synthetic pilots use hierarchic behaviors to implement portions of the tasks. That is, higher-level behaviors control the instantiation and sequential activation of lower-level behaviors for each pilot. To coordinate the activities between pilots, the designers sought to make sure that specific, selected team-level tasks are jointly selected by different agents. We extract several general cases of coordination:

Agreement. Two or more agents have to simultaneously and synchronously select a single plan for joint execution by the entire team [11,8]. In the scenario above, agreement was to be achieved on the team-level behaviors, e.g., f, j, w, and h.

Simultaneous role selection coordination. Two or more agents have to select from a pool of behaviors, such that when one agent has selected a specific behavior another agent has selected the appropriate corresponding behavior. In the scenario above, such coordination takes place when the scouting subteam members select l while the attackers select b.

Sequential coordination. Two or more agents select states in a particular coordinated sequence. For instance, when engaging the enemy, the helicopters may want to fire in particular sequence such that all helicopters go through $m \rightarrow u \rightarrow s \rightarrow m \rightarrow ...$ in the same order. When one helicopter is executing a particular step in this sequence, its teammate executes the next step in the sequence.

The problem is to verify that the individual selection of team-level behaviors is coordinated [11]. Under the limited communication range and reliability conditions in this domain, as well as security restrictions imposed on the pilots, an OBC scheme was required [8]. The key to this approach is for each pilot agent to observe its teammates using radar. Based on their velocity and altitude, the agent would infer what behaviors are being executed (a behavior-recognition process), and detect failures in coordination. For most cases, the same observation can be interpreted as being indicative of different behaviors. Hence, there can be multiple interpretations for the observation. The agent can then use an optimistic or pessimistic monitoring policy to select between these interpretations, thereby guaranteeing sound or complete detection quality (see [8]) for details).

We will now show how these examples can be formalized. We start by defining the basic elements of OBC before continuing to the failure detection (FD) policies and algorithms. To illustrate the formalism, we use the Simultaneous Role Selection coordination as described above.

3.2 Definitions

Consider a team of n agents. Each of the agents has several internal states that are in general unobservable to any other agent but itself. We define the state space to be identical for each agent. In practice agents may use only a subset of the full state space (i.e., different agents have different state spaces). The state space is then:

$$S = \{s_1, ..., s_m\} \tag{1}$$

In our working example, we define the state space to be $\{b, c, l, f, w, m, u, s, h\}$

As an agent's state is typically not directly observable, each agent must engage in behavior recognition. An agent's state must be inferred by observing its behavior. We formalize observations as discrete members of a set

$$O = \{o_1, ...o_k\} \tag{2}$$

Observations map to states in some predefined manner. We define the observation-state mapping, which we call the recognition function, for agent j to be:

$$\begin{gathered} M_j : O \rightarrow S' \in S = \{(o_1, S'_{j,1}), ... \left(o_k, S'_{j,k}\right)\} \\ \text{where } o_1, .., o_k \in O \text{ and } S'_{j,1}, .., S'_{j,k} \subseteq S, \text{ for } j = 1..n \end{gathered} \tag{3}$$

In our working example, observations and recognition function may be:

$$O = \{(o_1 : speed < 3), (o_2 : (altitude > 3) \wedge (speed > 3))\} \tag{4}$$

$$M_1 = M_2 = ... = M = \{\langle o_1, b\rangle, \langle o_1, l\rangle, \langle o_2, b\rangle\} \tag{5}$$

Intuitively, the agent slows down when waiting (b) and moves forward at altitude when locating the enemy's position (l). It may also slow down in l, making observation o_1 ambiguous.

Observations on the team of agents form a joint observation space defined as:

$$JO = O \times ... \times O = \{\langle o_{i_1}, ..., o_{i_n} \rangle \,|\, o_{i_j} \in O,\ \forall i_j = 1..k, j = 1..n\} \tag{6}$$

We will use the following notation for an element of the joint observation set.

$$jo_{i_1,...,i_n} = \langle o_{i_1}, ..., o_{i_n} \rangle \in JO \tag{7}$$

Each element of the joint observation set, via the recognition function, maps to one or more elements in a joint state space, formed by the cross products of the state spaces.

$$JS = S \times ... \times S = \{\langle s_{i_1}, .., s_{i_n} \rangle \,|\, s_{i_j} \in S,\ \forall i_j \in \{1..m\}, j = 1..n\} \tag{8}$$

We will use the following notation for an element of the joint states set.

$$js_{i_1,...,i_n} = \langle s_{i_1}, ..., s_{i_n} \rangle \in JS \tag{9}$$

For a given joint observation, $jo_{i_1,...,i_n}$, we call the set of possible joint states the team could be in, the observed joint states OJS or hypotheses set. We can construct OJS as the cross-product of the mapped sub-sets from M_j for each agent $j = 1..n$. That is for the given joint observation we form:

$$\begin{aligned} OJS_{i_1,...,i_n} &= M_1(o_{i_1}) \times ... \times M_n(o_{i_n}) \\ &= S'_{1,i_1} \times S'_{2,i_2} \times ... \times S'_{n,i_n} \end{aligned} \tag{10}$$

In our example, given a joint observation $\langle o_1, o_1, o_2 \rangle$, the observing agent may conclude that the OJS for the observed agent is: $\{\langle b, b, l \rangle, \langle b, l, l \rangle, \langle l, b, l \rangle, \langle l, l, l \rangle\}$. In order to detect failures in coordination, the hypotheses set is compared to the set of *desired* joint states (DJS) as specified by the designer. DJS, a subset of JS, is defined as:

$$DJS = \{js_{i_1,...,i_n} \,|\, js_{i_1,...,i_n} \in JS \wedge js_{i_1,...,i_n} \text{desired}\} \subseteq JS \tag{11}$$

In principle, if $DJS \cap OJS = \emptyset$ then the observed agents' joint state is definitely not desired and a clearly recognizable coordination failure has occured. In contrast, if $DJS \cap OJS = OJS$ then no failure occured. In general, however, DJS and OJS only partially overlap meaning some joint states are acceptable and some are not. Therefore, we must classify failures via some policy in these ambiguous cases.

We define two failure detection policies: π_{PFD} a pessimistic policy and π_{OFD} an optomistic policy. π_{PFD} takes any possibility of a failure to mean a failure occurred, while π_{OFD} reports failures only when a clear failure occurred. Mathematically, π_{PFD} reports failures when *any* element of OJS is not an elements of DJS. In contrast, π_{OFD} reports failures only when *no* element of OJS is an element of

DJS. It can be shown that π_{OFS} will never report a false-positive, while π_{PFS} will never report a false-negative. Thus:

$$\pi_{PFD}\left(jo_{i_1,\ldots,i_n}\right) = \begin{cases} \text{failure} & OJS_{i_1,\ldots,i_n} \not\subseteq DJS \\ \text{okay} & \text{otherwise} \end{cases} \tag{12}$$

$$\pi_{OFD}\left(jo_{i_1,\ldots,i_n}\right) = \begin{cases} \text{failure} & OJS_{i_1,\ldots,i_n} \cup DJS = \emptyset \\ \text{okay} & \text{otherwise} \end{cases} \tag{13}$$

3.3 Examples

To better clarify the formalism, let us consider a simple three agent system with three states s_1, s_2, s_3 and two observations o_1, o_2. Let us define the recognition functions and DJS as:

$$M_j = M = \{\langle o_1, s_1\rangle, \langle o_1, s_2\rangle, \langle o_2, s_3\rangle\} \text{ for } j = 1, 2, 3 \tag{14}$$

$$DJS = \{\langle s_1, s_1, s_1\rangle, \langle s_2, s_2, s_2\rangle, \langle s_3, s_3, s_3\rangle\} \tag{15}$$

Let us consider the three joint observations $\langle o_1, o_1, o_1\rangle$, $\langle o_1, o_1, o_2\rangle$ and $\langle o_2, o_2, o_2\rangle$. Thus we have:

$$jo_{1,1,1} \rightarrow M(o_1) \times M(o_1) \times M(o_1) = \{s_1, s_2\} \times \{s_1, s_2\} \times \{s_1, s_2\} \tag{16}$$

$$= \{\langle s_1, s_1, s_1\rangle, \langle s_1, s_1, s_2\rangle, ..., \langle s_2, s_2, s_2\rangle\} \tag{17}$$

$$jo_{1,1,2} \rightarrow \{\langle s_1, s_1, s_3\rangle, \langle s_1, s_2, s_3\rangle, \langle s_2, s_1, s_3\rangle, \langle s_2, s_2, s_3\rangle\} \tag{18}$$

$$jo_{2,2,2} \rightarrow \{\langle s_3, s_3, s_3\rangle\} \tag{19}$$

The first observation contains some elements that are also in DJS and so passes the optomistic FD policy but fails the pessimistic one. The second observation contains no elements from DJS and thus fails both policies and is a clear failure. The final observation is a strict subset of DJS and therefore is a success using either policy.

As a final example, let us return to our earlier helicopter example. Let us assume that the first agent is a scout and the other two agents are attackers. We have $DJS = \{\langle l, b, b\rangle\}$. Given a joint observation $\langle o_1, o_1, o_2\rangle$, OJS is $\{\langle b, b, l\rangle, \langle b, l, l\rangle, \langle l, b, l\rangle, \langle l, l, l\rangle\}$. The π_{PFD} policy would report a failure as $OJS \not\subseteq DJS$. π_{OFD} would also report a failure as $OJS \cap DJS = \emptyset$. In contrast, for $\langle o2, o1, o1\rangle$, π_{PFD} reports a failure while π_{OFS} does not. In general, the set of joint observations that are failures according to π_{OFD} are a subset of those classified as failures by π_{PFD}.

4 Compiling Reactive Monitoring Rules

Performing the failure detection (FD) policy evaluation on-line becomes intractable in many problems, particularly where computational resources are at a premium. The core computational cost in the policy evaluation occurs in the OJS calculations.

The calculation for OJS, see 10, are exponential in the number of agents. Off-line compilation of reactive rules offers one method for addressing this issue.

To compile reactive rules, we want to generate from our apriori knowledge a mapping from JO to the two-element space, $\{fail, pass\}$, representing the output of the policy evaluation. Our approach divides JO into two subsets, one containing joint observations that map to failures FJO, and the other containing joint observations that map to successes SJO. Thus we define FJO and SJO for failure detection policy π_{FD} as:

$$FJO = \{jo_{i_1,\ldots,i_n} | jo_{i_1,\ldots,i_n} \in JO \wedge \pi_{FD}\left(jo_{i_1,\ldots,i_n}\right) = \text{failure}\} \subseteq JO \quad (20)$$

$$SJO = \{jo_{i_1,\ldots,i_n} | jo_{i_1,\ldots,i_n} \in JO \wedge \pi_{FD}\left(jo_{i_1,\ldots,i_n}\right) = \text{success}\} \subseteq JO \quad (21)$$

FJO and SJO partition JO such such that $FJO, SJO \subseteq JO$ and $FJO \cup SJO = JO$. For the ensuing discussion we will use the non-failure set SJO with the optimistic π_{OFD} policy. A similar approach could be taken for the π_{PDF} policy but will not be discussed here.

It is readily apparent that building SJO by testing each individual element of OJS with the policy function, π_{OFD}, will take an exponential amount of time. Clearly, a better approach is required. Our approach operates by recognizing that we only need to find those joint observations that have any corresponding joint states in DJS. Thus, for each joint state in DJS we generate the set of observation tuples that could possibly map to that joint state. For a given element of a joint state that is in DJS the $GENO$ function, short for GENerate Observations, will perform this tasks as:

$$GENO\left(j, js_{i_1,\ldots,i_n}\right) = \{o_v | s_{ij} \in \bigcup_{\alpha=1..N} M_\alpha\left(o_v\right)\} \ \ where \ \ js_{i_1,..i_j..,i_n} \in DJS \quad (22)$$

Generating the set of joint observations that could possibly map to the given joint state is a cross product operation as:

$$GENJO\left(js_{i_1,..,i_n}\right) = GENO\left(1, js_{i_1,..,i_n}\right) \times .. \times GENO\left(n, js_{i_1,..,i_n}\right) \quad (23)$$

The complete compilation operates over all the elements in DJS with the results combined with the union operator to form SJO as:

$$SJO\left(DJS\right) = \bigcup_{js_{i_1,\ldots,i_n} \in DJS} GENJO\left(js_{i_1,\ldots,i_n}\right) \quad (24)$$

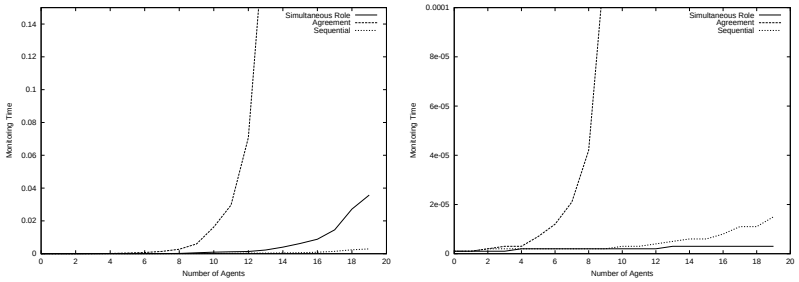
5 Evaluation

We implemented a system that accepts monitoring examples written in the presented formalism. The system performs monitoring when given observation tuples using both uncompiled and compiled mechanisms. To evaluate the performance of compilation, we translated all three examples described in section 3 using the formalism and compared the mean execution times for 100 randomly selected observation

tuples. Trials were performed with between 1 and 20 agents using the optomistic failure detection policy.

Figure 1 shows the average running times for the compiled and non-compiled versions. The X axis shows the number of agents in each configuration. The Y axis measure monitoring time in seconds. Figure 1-a shows that the run-time curve as the number of agents increase is non-linear in all three examples, indeed clearly exponential in the case of the agreement coordination example (in which the observations lead to very ambiguous interpretations).

The compiled monitoring results (Figure 1-b) follow very similar trends to those of the non-compiled versions, however their running times are *orders of magnitude* smaller (note the difference in range on the Y axis between the two figures). While the curves for the compiled "Simultaneous Role" and "Sequential" coordination monitoring seem almost linear, the curve for compiled agreement monitoring again grows exponentially, though much slower than its non-compiled version. As described in Section 4, the compiled rules may still require exponential time in the worst case.



(a) Un-compiled (b) Compiled

Fig. 1. Average running times for the non-compiled and compiled monitoring. *Note the Y-axis range is significantly different in Figures a and b.*

6 Conclusions and Future Work

This paper introduces a novel formalism for describing observation-based monitoring for coordination failure, an important component in observation-based coordination. The formalism allows application-independent investigation of the monitoring process, and facilitates analytical treatment of monitoring. We provide an example of such analytical treatment, a process that compiles the exponential run-time process of monitoring into a set of reactive rules that can be executed quickly. We evaluate our work empirically on varying-scale examples of observation-based

monitoring taken from the literature, and show that the compilation process results in orders-of-magnitude faster run-time. Future directions for our work include in-depth study of the compilation process and the effects of ambiguous observations on monitoring run-time.

Acknowledgements This research was sponsored by the United States Air Force and the United States Army under Cooperative Agreements Nos F30602-00-2-0549, F30602-98-2-0135 and DABT63-99-1-0013. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force, the Navy, or the US Government.

We thank Guy Lebanon and Yuval Kaminka for useful discussions.

References

1. Milind Tambe, W. Lewis Johnson, Randy Jones, Frank Koss, John E. Laird, Paul S. Rosenbloom, and Karl Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), Spring 1995.
2. Marcus James Huber and Tedd Hadley. Multiple roles, multiple teams, dynamic environment: Autonomous netrek agents. In W. Lewis Johnson, editor, *Proceedings of the International Conference on Autonomous Agents*, pages 332–339, Marina del Rey, CA, 1997. ACM Press.
3. Hiroaki Kitano, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, E. Osawa, H. Matsubara, Itsuki Noda, and M. Asada. The RoboCup synthetic agent challenge '97. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997.
4. Lynne E. Parker. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.
5. Tucker Balch. *Behavioral Diversity in Learning Robot Teams*. PhD thesis, Georgia Institute of Technology, 1998.
6. Maja J. Mataric. *Interaction and Intelligent Behavior*. PhD thesis, Massachusetts Institute of Technology, 1994.
7. Yasuo Kuniyoshi, Sebastien Rougeaux, Makoto Ishii, Nobuyuki Kita, Shigeyuki Sakane, and Masayoshi Kakikura. Cooperation by observation—the framework and the basic task patterns. In *the IEEE International Conference on Robotics and Automation*, pages 767–773, San-Diego, CA, May 1994. IEEE Computer Society Press.
8. Gal A. Kaminka and Milind Tambe. Robust multi-agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research*, 12:105–147, 2000.
9. Richard Washington. Markov tracking for agent coordination. In *Proceedings of the International Conference on Autonomous Agents*, pages 70–77, Minneapolis/St. Paul, MN, 1998. ACM Press.
10. Milind Tambe. Tracking dynamic team activity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, August 1996.
11. Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.