In *Proceedings of the Eighth Conference on Intelligent Autonomous Systems, March, 2004*

Towards a Comprehensive Framework for Teamwork in Behavior-Based Robots

Gal A. Kaminka, Yehuda Elmaliach, Inna Frenkel,
Ruti Glick, Meir Kalech, Tom Shpigelman
The MAVERICK Group
Computer Science Department
Bar Ilan University, Israel
galk@cs.biu.ac.il

Abstract. Teams of robots are increasingly deployed in real world applications. One of the key challenges in building such teams is to automate the control of *teamwork*, such that the designer can concentrate her efforts on the *taskwork* to be done. This paper presents steps towards a novel framework addressing this challenge in teams of behavior-based agents. The framework provides a rich representation that facilitates management of teamwork knowledge, and separates behaviors that govern a robot's interaction with its task from behaviors that govern a robot's interaction with its teammates. In addition, the paper presents a set of algorithms that control the execution and communication of behaviors, to automate synchronization and allocation of behaviors to sub-teams. We describe our implementation of the framework in the BITE architecture, a distributed behavior-based architecture providing automated coordination and collaboration services in a team of Sony AIBO robots.

1 Introduction

Teamwork in autonomous agents and robots is fast becoming an area of significant interest to academic and industrial research groups, motivated by interest in deploying autonomous and semi-autonomous teams of agents in real-world applications. Increasingly, algorithms and associated representations are being developed to automate the interactions between teammembers, in order to facilitate robust deployment. Indeed, such algorithms have already been used with robots and virtual robots to address important aspects of teamwork: synchronization [9], task allocation [7, 12, 2], proactive communications [13], teamwork failure detection, diagnosis and repair [9, 4, 3], anticipation of teammate needs [11], and human-in-the-loop [8]. Unfortunately, to the best of our knowledge, no attempt has been made to synthesize these important contributions into one coherent algorithmic framework. As a result, key questions in teamwork remain open.

This paper presents our first steps towards a comprehensive framework for controlling teams of behavior-based agents and robots, synthesizing and integrating many existing algorithms, and adding novel capabilities. The framework relies on maintaining and linking three graph structures: An organization hierarchy, a task/sub-task behavior hierarchy, and a set of social behaviors. The social behaviors handle interactions between the agents, and are triggered upon successful or failed termination of the individual behaviors of agents. The key idea in the separation of these three components is to simplify team control algorithms

by decoupling control of individual behavior execution from the control of team-members' interactions. Indeed, we show various teamwork algorithms that operate on these structures.

We describe BITE (*Bar Ilan Teamwork Engine*), an implementation of these algorithms in a distributed control architecture. BITE provides several automated teamwork services to behavior-based robots, and currently targets the Sony AIBO platforms, with additional domains on the way. We demonstrate BITE's capabilities in running formation-following tasks, and show that it automatically enables coordinated activity without burdening the designer with additional work.

This paper is organized as follows. Section 2 provides a brief survey of existing work and motivates the framework. Section 3 presents the framework. Section 4 presents our implementation of the framework in the BITE architecture, and results of experiments run with BITE. Section 5 concludes.

2 Motivation and Background

Our work was inspired in particular by four investigations that have tackled teamwork at an architectural level. Tambe's work on the STEAM teamwork model [9] have explored the use of automated decision-theoretic communications in synchronizing the selection and termination of hierarchical behaviors, and also the use of organizational repair behaviors that could be triggered based on failures in the organizational structure (such as a teammate permanently crashing). Parker's work on the ALLIANCE architecture [7] has investigated a robust distributed behavior-based control architecture in which robots dynamically allocated and re-allocated themselves to tasks, based on their own sensed failures and those of their teammates. Yen et al.'s CAST architecture [13] demonstrated a mechanism for proactive communications, in which virtual robots anticipate the information needs of their teammates and respond appropriately. SCORE [12] demonstrated the usefulness of using different protocols within a single architecture.

All of these architectures are *monolithic*. They provided some capabilities but not others, and do not allow the designer to add or subtract capabilities: STEAM provided synchronization and some allocation services, but uses a fixed protocol to achieve coordinated activity. SCORE uses multiple protocols, but lacks many of the fault-tolerance capabilities of STEAM and ALLIANCE. ALLIANCE is extremely robust to failures, but does not explicitly synchronize robots as they jointly take on tasks. CAST provides proactive communications, but does not provide many of the services in STEAM.

To rapidly deploy robotic teams, we believe that a comprehensive framework is needed which integrates many of these capabilities using representations and algorithms for teamwork. However, this integration must not be monolithic, in the sense that it must allow the designer to explore alternative protocols and automated coordination capabilities, for the task at hand. The framework we provide seeks to fulfill this vision. It can provide many, if not all, of the capabilities of previous investigations, but teases apart coordination, control, and communications. None of the previous investigations allows such separation, which we achieve through the maintenance of separate social behaviors. Thus for instance, it is possible in our framework to switch between multiple synchronization methods, to dynamically re-allocate robots to tasks in more than one way, and to manage proactive communications. However, our work still lacks the failure-recovery facilities of STEAM or ALLIANCE.

Additional recent work in multi-robot systems underscores the need for a comprehensive framework that facilitates automation of teamwork services. Goldberg et al. explore a dis-

tributed architectures based on the traditional three-tier architecture, in which multiple robots interact with each other at all three layers [2]. A key difference between this work and ours is that Goldberg et al. makes a commitment to a market-based resource allocation scheme, while we leave the allocation method in the hands of the designer. The designer may allow robots to use a market-based approach, but may also direct them to using other methods.

3 A Framework for Teamwork

The framework uses hierarchical behaviors as the basis for a representation underlying the controllers for the team members. We add two additional structures to this representation: a set of social behaviors, and an associated team-hierarchy (described in Section 3.1). We then describe a number of principal algorithms that use this representation to automate control and communications of a team of robots (Section 3.2).

3.1 Representation

The first of the three structures specifies the sequential and hierarchical relationships between behaviors that execute the task. The *task behavior graph* is an augmented connected graph tuple $< B, S, V, b_0 >$, where B is a set of task-achieving behaviors (as vertices), S, V sets of directed edges between behaviors ($S \cap V =$), and $b_0 \in B$ a behavior in which execution begins. Each behavior in B may have preconditions which enable its selection (the robot can select between enabled behaviors, subject to the constraints described below), and termination conditions that determine when its execution must be stopped. S is a set of *sequential* edges, which specify temporal order of execution of behaviors. A sequence of behaviors that follows sequential edge is called an *execution chain*. A sequential edge from b_1 to b_2 specifies that $b1$ must be executed before executing $b2$. V is a set of vertical *task-decomposition* edges, which allow a single higher-level behavior to be broken down into execution chains containing multiple lower-level behaviors. At any given moment, the robot is executing a complete path—root-to-leaf—through the behavior graph. Task-behavior graphs such as this (with some variations) are fairly popular in robotics and multi-agent systems [1, 7, 6]. We impose several structural constraint on the behavior graph. The first is straightforward: sequential edges may form circles, but vertical edges cannot be a part of a circle. This allows an execution chain to be repeated by choice, but does not allow hierarchical recursion where a behavior is decomposed into executing itself.

However, we impose an additional constraint on the semantics of multiple outgoing edges. Two outgoing sequential edges $< a, b >, < a, c >$ signify a choice point between *alternative* execution chains: either b or c must be selected by the robot once its execution of a is finished. In contrast, two outgoing decomposition edges $< a, b >, < a, c >$ signify *complementary* execution chains: Both the execution chain beginning with b and the execution chain beginning with c must terminate for a to be considered complete. By convention, vertical edges in this case point only to the first behaviors of execution chains–since in any case such behaviors must be executed before others in their respective chains. The motivation for these two constraints will be discussed below.

Figure 1-b shows an example of a simple behavior graph, constructed for experimenting with multi-robot formation maintenance tasks. Here, there are two formation behaviors—*triangle formation* and *line formation*. Execution begins with triangle formation, and can (under specific conditions) switch to the line formation. Both formations use one behavior–

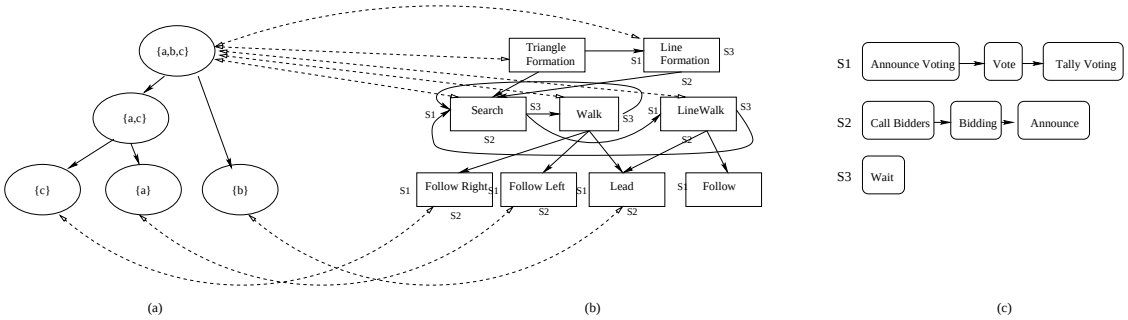

Figure 1: Portions of the team hierarchy (a), behavior graph (b), and social behaviors (c) for a formation-maintenance task. Links from (b) to social behaviors in (c) are denoted by S1–S3.

search–which causes robots to localize themselves with respect to others. Once localized, the robots choose between the *walk* behavior (which implements walking in triangle) or the *linewalk* behavior in which robots follow each other in a line. Selection between these options must be synchronized across robots—as described below.

Behavior graphs (e.g., [1, 6]) have traditionally been used to proscribe individual behaviors, i.e., intra-robot control. Their use in teams of robots is seemingly straightforward: Each robot has its own copy of the behavior graph. Behaviors whose selection must be coordinated across the team (i.e., behaviors responsible for inter-robot control) issue appropriate communication actions to coordinate execution. Unfortunately, it was previously shown [9, 4] that this simplicity is deceptive, as this approach breaks down in complex, dynamic environments, and when the organizational structure of the team becomes more complex .

The two remaining structures directly represent organizational structure and organizational behavior, which seek to address these difficulties. The first of these is the organization hierarchy, also called the team hierarchy in [4, 10]. This is a DAG (Directed Acyclic Graph) whose vertices are associated with sub-teams of agents, and whose edges signify sub-team-membership relationships. Several vertices appear in any organization hierarchy: Given the complete set of robot team-members R, a vertex corresponding to R (and representing the entire organization) is a part of the hierarchy, as are all the singleton sets $\{r_i\}$, where $r_i \in R$. Other vertices correspond to multi-robot sub-teams of robots in R and are connected such that if there exists an edge $< R_1, R_2 >$, then $R_2 \subset R_1$. The team hierarchy thus forms a partial lattice, from the root team R which includes all team-members, to sub-teams corresponding to each of the members by itself (i.e., to the individuals in the organization).

To allow behaviors to reason about the organization responsible for their execution, we create links between the behavior graph and the team hierarchy, such that there is a link from a behavior B_j to a sub-team R_i if B_j is to be jointly executed by R_i. Similarly, to allow reasoning about allocated tasks, we link sub-teams to the behaviors they are responsible for, such that there's a link from a sub-team R_i to behavior(s) B_j if R_i is responsible for B_j. For instance, in Figure 1-a, one can see a team hierarchy composed of three robots, simply identified as a, b and c. Each team is linked with the behaviors associated with it, and the behaviors are linked with their associated teams when these are known (shown in the figure as bi-directional links).

Using these links between the behavior graph and the team hierarchy, a robot executing a behavior may easily find out whom it should contact in order to coordinate execution of this behavior. However, its actions to achieve this coordination remain unspecified. For instance,

suppose three robots are executing the formation task triangle formation (Figure 1-b) have together finished execution of the behavior *search*, and have started on *walk*. We remind the reader that we impose a semantic constraint whereby multiple decomposition edges signify an allocation choice. The robots must jointly decide how to allocate the different roles of the formation between them. One must lead the triangle at the front (the *lead* behavior), while the others follow—one from the left (*follow left*) and the other from the right (*follow right*). To negotiate this synchronized decision, the robots may communicate, for instance by executing a bidding protocol where different robots bid on the behaviors they wish to execute. Once this decision is made, links are created from each behavior to the appropriate vertices in the team-hierarchy, to denote who is executing what.

True to the behavior-based approach, we allow the designer to define *social behaviors*, i.e., control modules which address strictly inter-agent aspects of behavior, such as the voting behavior previously described. These behaviors are collected together in the third structure, which contains a set of (possibly unconnected) behavior graphs—each corresponding to a social behavior—that may have decomposition and sequential transitions leading to other social behaviors. In order to facilitate the selection and execution of social behaviors in appropriate points in execution, we link the task behaviors to social behaviors in three separate ways: (a) synchronized selection of behaviors prior to their execution; (b) team-wide allocation of robots and sub-teams to behaviors; and (c) synchronized termination of behavior execution. Social behaviors typically control communication actions and execute interaction protocols (e.g., voting) that govern coordinated activity.

Synchronized selection occurs when new team behaviors are selected for execution, in particular when a decision is to be made between several sequential transitions. For instance, in Figure 1-b, two sequential transitions leave the behavior *Search*—one going into the behavior *Walk*, and one going into the behavior *LineWalk*. A synchronized decision is to be made between these (such that all robots select the same behavior), and execution must begin simultaneously. An appropriate social behavior is used to coordinate this synchronized selection. For instance, Figure 1-c shows a simple voting behavior (marked $S1$) in which one pre-determined robot announces the call for votes and the candidate behaviors, then collects the votes by all team members and announces the winning behavior. This behavior is then selected for execution by each robot.

Allocation of sub-teams to behaviors occurs when a behavior is to be decomposed into children behaviors. If only one decomposition transition exists, then the entire team selects it. Otherwise, if multiple decomposition transitions exist, then this is taken to mean that the team is to be split into sub-teams. The appropriate social behavior is called to carry out this allocation, for instance by using a market-based approach [2], or an agenda-based mechanism [7] to divide up the work among robots. In Figure 1-c, behavior $S2$ marks the sequential phases of a market-based protocol for use in allocating the children behaviors to different sub-teams. Once this allocation is made, appropriate links are created between the allocated behaviors in the behavior-graph and the sub-teams in the team-hierarchy responsible for executing them.

Finally, **synchronized termination of behavior execution** determines the social behavior of robots as they reach the end of an execution chain. Normally, upon terminating an execution chain, control is passed back to the parent behavior, which is then also terminated. However, if a parent behavior is associated with a sub-team composed of several members, then termination of the execution chain must be coordinated, so that teammates know that it is done with its allocated execution chain. For instance, if the parent behavior has several robots doing a distributed search for a target, then the first robot to find the target will neces-

sarily want to terminate the search and inform its teammates. To control this social behavior, a synchronized termination behavior is called. In Figure 1-c, behavior $S3$ marks a very simple synchronized termination behavior which is appropriate for the formation task. In the behavior *Wait*, a robot that has terminated execution of a joint behavior waits for all other robots to reach the end of their execution chains as well, before they all begin their joint execution of a new behavior. Alternatively, an agenda-based synchronization mechanism could have been used to assign the robot, now free of its previously-allocated behavior, to a new behavior.

3.2 Teamwork Algorithms

We now describe some of the principal algorithms that use the representation above. We begin with the basic algorithm that controls the coordinated execution and selection of behaviors by robots. As previously described, the control loop executes *a behavior stack*—root behavior to leaf—where all behaviors on the stack are executed simultaneously with their currently selected children.

Each of the robots executes Algorithm 1. Execution begin with putting the initial behavior of the behavior graph (typically the root) on the execution stack (lines 1–2). Then the algorithm essentially loops over four phases in order. First, it recursively expands the children of the behavior, allocating them to sub-teams if necessary (lines 3a–3c). It then executes the behavior stack in parallel, waiting for the first behavior to announce termination (lines 4a–4c). All descendants of a terminating behavior are popped off the stack (i.e., their execution is also terminated—line 4b), and then a synchronized termination takes place (line 6). This can result in a newly-allocated behavior within the current parent context, in which case, it will be put on the stack for expansion (line 7). Otherwise, this indicates that the robot should select between any enabled sequential transitions from the terminated behavior (lines 8a–8e). This process normally results in new behaviors put on the stack. Thus a final goto (line 9) back to line 3 begins again with their recursive expansion and allocation to sub-teams.

The recursive allocation of children behaviors to sub-teams in lines 3a–3c relies on the call to the $Allocate()$ procedure. It takes the current execution context (i.e., current stack, available children), and then first checks in the team hierarchy (via the links from the parent behavior to the team hierarchy) whether the children are already allocated sub-teams (e.g., by the designer). If this allocation is complete (all sub-teams have assignments) and correct (e.g., no sub-team is assigned to two different behaviors), then this allocation is taken to be predefined by the designer, and transmitted to all other robots on the sub-team responsible for the current parent (e.g., behavior on top of stack), until all members are in agreement about how to split the parent's sub-team into smaller sub-teams that are responsible for the children. If the allocation is lacking in some way, the appropriate social behavior in O, (linked from the current parent) is called to execute a procedure that will make this decision. Any current allocation can then be used to guide the selection. The current execution stack is used to help guide allocations—for instance by conveying information about where in the behavior graph the allocation is taking place.

Once a final allocation is determined, $Allocate()$ is responsible for updating the links from the behavior graph to the team hierarchy (and vice versa) to reflect the allocation. It then returns, for each robot, the child behavior for which it is responsible as part of the split sub-team (or individually, if the sub-team is composed only of the individual robot).

Synchronized termination (line 5–7) and selection (lines 8a–8e) similarly rely on calls to the procedures $Terminate()$ and $Decide()$, respectively. $Terminate()$ is responsible for evoking the execution termination social behavior, which can return a new child behavior for

Algorithm 1 $Control$(behavior graph $< B, S, V, b_o >$, team hierarchy T, social behaviors O)

1. $s_0 \leftarrow b_0$ // initial behavior for execution
2. push s_0 onto a new behavior stack G
3. Let $A \leftarrow \{b_i | < s_0, b_i >$ is a decomposition transition in $V\}$ // Allocate sub-task behaviors
   (a) if A has only one behavior b, $push(G, b)$.
   (b) else $b \leftarrow Allocate(G, s_0, A, T, O)$, then $push(G, b)$.
   (c) $s_0 \leftarrow b$.
4. execute in parallel for all behaviors b_i on G: // Execution
   (a) execute b_i until it terminates
   (b) while $b_i \neq top(G)$, $pop(G)$
   (c) break parallel execution, goto 5.
5. $b \leftarrow pop(G)$ // Terminate joint execution:
6. $c \leftarrow Terminate(G, b, T, O)$
7. if $c \neq NIL,\ push(G, c)$
8. else: // Select next behavior in execution chain
   (a) Let $Q \leftarrow \{s_i | < b_0, s_i >$ is a sequential transition in $S\}$
   (b) if Q is empty, goto 5 // terminate parent
   (c) if Q has one element s, $push(G, s)$
   (d) else $s \leftarrow Decide(G, b_0, Q, T, O)$
   (e) $s_0 \leftarrow s$
9. If G not empty, goto 3.

execution under the current parent. If it doesn't, then the next behavior in the execution chain must be decided on by $Decide()$, which calls the appropriate social behavior. Since synchronized selection involves all members of the current sub-teams selecting together, this social behavior would communicate with the members of the sub-team assigned to the terminated behavior. Note that in step 8b we also handle the case where no more behaviors are available in the execution chain. This case signals a termination of an execution chain, which in turn signals termination of the parent, thus the branching back to line 5. We omitted here the obviously needed check on whether a parent actually exists—if not, then the end of the behavior graph has been reached, and execution halts.

Algorithm 2 presents provides the team with the capability to inform teammates of sensed knowledge that is relevant to them. The algorithm is intended to run concurrently with the control algorithm described above. It works in two phases. In the first phase (*obligatory* communications—lines 1a–1c), each robot determines whether new information which affects its current behavior stack has become available, such as newly-satisfied conditions. These potentially affect the robot's immediate teammates, and must therefore be communicated to them by finding out which sub-team is responsible for each behavior on the stack. The second phase (proactive communications—lines 2a–2b) allows the robot to determine whether newly sensed information may be relevant to a sub-teams that it is not a member of, providing them with information even though it is not strictly its own responsibility to do so (in the sense that it is not its own sub-team that requires this information).

For instance, suppose a team of robots is executing the formation task described above, with the sub-team allocations as described in Figure 1. Suppose that the robots are currently executing the *Triangle Formation* behavior, and are currently executing the *Walk* behavior.

Algorithm 2 $Communicate$(behavior graph $< B, S, V, b_0 >$, team hierarchy T)

1. for all behaviors b on behavior stack G: // obligatory communications
   (a) $t \leftarrow subteam(b)$ // sub-team responsible for behavior b
   (b) if a termination condition of b is satisfied, inform all members of t
   (c) if a precondition of a sequentially-next behavior f ($< b, f >\in S$) is satisfied, inform all members of t
2. for all teams t in the team hierarchy: // proactive communications
   (a) $C \leftarrow \{b | b \in B, t$ currently linking to $b\}$
   (b) for all $b \in C$ and not on the behavior stack:
      i. if a termination condition of b is satisfied, inform all members of t
      ii. if a precondition of a sequentially-next behavior f ($< b, f >\in S$) is satisfied, inform all members of t

Suppose now that one of the follower robots is currently executing the *Follow Left* behavior. Phase 1 of Algorithm 2 guarantees that if this robot discovers that any of the termination conditions of Follow Left, Walk, or Triangle Formation, then it will inform the appropriate members of its team. Phase 2 guarantees that if the robot discovers a termination condition for *Lead*, then it will inform the members of the sub-team associated with *Lead*, even though they are not members of the same sub-team.

Additional algorithms can be derived based on analysis of the three structures and their interacting links. For instance, it is fairly straightforward to adapt a behavior-recognition algorithm such as RESL [4, 3] to recognize what behaviors other robots are executing, based on their observable actions. This information can be very useful to monitor for coordination failures that can occur if communication fails, and to focus communications only on what others do not yet know. Similarly, straightforward analysis of the behavior graph can yield anticipatory information about which behaviors are expected to be selected, thus allowing robots to anticipate the needs of their teammates [11].

Finally, it is possible to extend the control algorithm to allow human-in-the-loop control or guidance. This can be done in several ways. First, the social behaviors that govern selection and allocation may appeal to human control. In other words, rather than executing a voting protocol, a synchronization social behavior may consult the human operator on which behavior the team should select. Second, the operator may pre-specify allocation of teams to behaviors before execution. These will be reflected in the results of the $Allocate()$ procedure as previously described. Third, when an operator takes over the control of a single behavior (in other words, controlling the robot rather than letting the behavior doing it), the other robots will still perceive it as acting as part of the team, and will therefore continue to communicate with it and coordinate with it seamlessly.

4 BITE: An Instantiated Teamwork Architecture

The previous section argues for the benefits of the presented teamwork framework on the basis of the capabilities it enables. It automates management of communications, both communications obligatory to teammates on the same sub-team, as well as communications anticipating information needs of teammates. It allows a robot team to easily manage sub-teams and therefore to carefully control which information should be shared with whom. It teases apart the coordination of behaviors from the execution of the same behaviors, thus facilitating coordination mechanism re-use in a behavior-based task representation.

(a) Successful triangle formation

(b) Failure to maintain triangle formation. The top left robot has gotten too close.

Figure 2: Successful and failing triangle formations, by Sony AIBO robots executing the BITE architecture.

Other aspects of the framework may be evaluated by using it on actual robot teams. For this purpose, we have developed the BITE distributed team control architecture. BITE is an implemented instantiation of the framework that currently targets the Sony AIBO family of robots (a port for use in the GameBots domain [5] is on its way). We use BITE to manage and automate the coordination of robots executing formation-maintenance tasks (Figure 2).

The formation behaviors themselves are built using simplistic algorithms, in which robots maintain relative angles and positions to a lead robot. Color segmentation is used to identify the angle to the lead robot, while each AIBO's sole distance sensor (infra-red) is used to maintain distance within some constraints.

Without a coordination mechanism, a simplistic algorithm like this is susceptible to sensing failures. For example, if one of the follower robots loses track of the lead robot, it may be left behind while the lead and the other follower robots move on. This can be fixed in principle by having the designer of the behaviors also cover this special case in the behavior itself. This, of course, requires the designer to anticipate this possible problem.

However, by implementing the behaviors in BITE, the burden of worrying about coordination is put on the robots. When one of the loses track of the lead robot, the appropriate termination condition is satisfied, and the robot switches to the *Search* behavior. Here BITE is automatically triggered. Algorithm 2 automatically transmits the change in the termination condition to the other teammates, thus causing them to stop as well, and also switch to the *Search* behavior. This has two benefits: First, all the robots stop and start together. Second, now that the other robots switch to the *Search* behavior, they in effect assist the failing robot in identifying the leader. In fact, depending on whether this is a precondition for the *Walk* behavior, the same Algorithm 2 may cause a robot that identifies the leader to automatically transmit its finding to other robots who may still be searching for it.

Indeed, as with other behavior-selection approaches, the design of the preconditions and termination conditions is key to a successful application. Behavior designs that do not make their conditions explicit will not be able to fully utilize BITE's coordination services. For instance, suppose a follower robot gets too close to the lead robot, due to distance-measurement errors (e.g., Figure 2-b). If the *Follow Left* and *Follow Right* behaviors treat this condition internally, and thus when a robot is to slow down, this is done without selecting a different behavior. However, an alternative design would have made this failure an explicit termination

condition of the *Follow* behaviors. In this case, the other robots would have been automatically informed of this by BITE, and would have been able to take recovery actions on their own (such as speeding up). However, the design of behaviors is a complex topic outside the scope of this paper.

5 Summary and Future Work

A key challenge in building robot teams is to automate teamwork, such that the designer can focus on planning the robots' taskwork. This paper presents a representation that enables such automation for behavior-based robots by separating behaviors that control social interactions from those that manage subtasks, and further distinguishing knowledge of the organizational structure. We present algorithms for controlling social interactions and communications using this representation, and an implementation of this framework in BITE, a distributed teamwork architecture that targets Sony AIBO robots in our lab. We plan to focus our efforts in the future on human-team interactions within the framework we presented.

Acknowledgments. We thank Avi Rosenfeld, Noam Hazon, and the anonymous reviewers for useful comments. As always, we thank K. Ushi for her support.

References

[1] R. James Firby. An investigation into reactive planning in complex domains. In *AAAI-87*, 1987.

[2] Dani Goldberg, Vincent Cicirello, M. Bernadine Dias, Reid Simmons, Stephen Smith, and Anthony Stentz. Market-based multi-robot planning in a distributed layered architecture. In *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings from the 2003 International Workshop on Multi-Robot Systems*, volume 2, pages 27–38. Kluwer Academic Publishers, 2003.

[3] Meir Kalech and Gal A. Kaminka. On the design of social diagnosis algorithms for multi-agent teams. In *IJCAI-03*, 2003. socially-attentive monitoring, diagnosis, plan-recognition, belief ascription.

[4] Gal A. Kaminka and Milind Tambe. Robust multi-agent teams via socially-attentive monitoring. *JAIR*, 12:105–147, 2000.

[5] Gal A. Kaminka, Manuela M. Veloso, Steve Schaffer, Chris Sollitto, Rogelio Adobbati, Andrew N. Marshall, Andrew Scholer, and Sheila Tejada. GameBots: A flexible test bed for multiagent team research. *Communications of the ACM*, 45(1):43–45, January 2002.

[6] Monica Nicolescu and Maja J. Mataric. A hierarchical architecture for behavior-based robots. In *AAMAS-02*, pages 227–233, Bologna, Italy, July 15–19 2002.

[7] Lynne E. Parker. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

[8] Paul Scerri, Lewis Johnson, David Pynadath, Paul Rosenbloom, Mei Si, Nathan Schurr, and Milind Tambe. A prototype infrastructure for distributed robot-agent-person teams. In *AAMAS-03*, 2003.

[9] Milind Tambe. Towards flexible teamwork. *JAIR*, 7:83–124, 1997.

[10] Milind Tambe, David V. Pynadath, Nicholas Chauvat, Abhimanyu Das, and Gal A. Kaminka. Adaptive agent integration architectures for heterogeneous team members. In *ICMAS-00*, pages 301–308, Boston, MA, 2000.

[11] Manuela Veloso, Peter Stone, and Michael Bowling. Anticipation: A key for collaboration in a team of agents. In *SPIE Sensor Fusion and Decentralized Control in Robotic Systems II (SPIE-99)*, 1999.

[12] Thuc D. Vu, Jared Go, Gal A. Kaminka, Manuela M. Veloso, and Brett Browning. MONAD: A flexible architecture for multi-agent control. In *AAMAS-03*, page In press, 2003.

[13] John Yen, Jianwen Yin, Thomas R. Ioerger, Michael S. Miller, Dianxiang Xu, and Ricahrd A. Volz. CAST: Collaborative agents for simulating teamwork. In *IJCAI-01*, pages 1135–1144, 2001.