Flexible Teamwork in Behavior-Based Robots

Gal A. Kaminka[*] and **Inna Frenkel**
The MAVERICK Group
Computer Science Department
Bar Ilan University, Israel
{galk,frenkei1}@cs.biu.ac.il

Abstract

A key challenge in deploying teams of robots in real-world applications is to automate the control of teamwork, such that the designer can focus on the taskwork. Existing teamwork architectures seeking to address this challenge are monolithic, in that they commit to interaction protocols at the architectural level, and do not allow the designer to mix and match protocols for a given task. We present BITE, a behavior-based teamwork architecture that automates collaboration in physical robots, in a distributed fashion. BITE separates task behaviors that control a robot's interaction with its task, from interaction behaviors that control a robot's interaction with its teammates. This distinction provides for flexibility and modularity in terms of the interactions used by teammates to collaborate effectively. It also allows BITE to synthesize and significantly extend existing teamwork architectures. BITE also incorporates key lessons learned in applying multi-agent teamwork architectures in physical robot teams. We present empirical results from experiments with teams of Sony AIBO robots executing BITE, and discuss the lessons learned.

Introduction

Teamwork in autonomous robots is fast gaining interest in academic and industrial research groups, motivated by real-world applications for multi-robot teams. To facilitate robust and speedy deployment of such teams, teamwork architectures are increasingly used to automate the interactions between team-members, such as synchronized task execution (Pynadath & Tambe 2003), and task allocation (Parker 1998; Dias & Stentz 2000; Gerkey & Matarić 2004; Vu *et al.* 2003). This allows the designer to focus on developing the taskwork, rather than the teamwork.

However, existing architectures leave important challenges open when applied to multi-robot teams. First, existing robot teamwork architectures do not provide both synchronized task execution and dynamic task allocation in the same architecture. Thus the team's developer must make a choice as to whether synchronization or allocation is more important. Second, existing architectures are monolithic, in the sense that they commit to a single interaction protocol for implementing each service, e.g., confirm-request for synchronization (Pynadath & Tambe 2003). Yet, as we show in our experiments below, the ability to flexibly mix and match protocols can critically impact task performance.

This paper argues that flexible teamwork must rely on a *micro-kernel* approach, in which different coordination protocols (such as synchronization and allocation) can be interchanged, even within the same task. For example, flexible teamwork must allow for robots to sometimes allocate tasks using market approaches (Dias & Stentz 2000), and sometimes based on robustness concerns (Parker 1998).

To make this argument concrete, this paper presents BITE (*Bar Ilan Teamwork Engine*), a novel behavior-based teamwork architecture targeting physical robot applications. Similarly to previous architectures (Pynadath & Tambe 2003; Vu *et al.* 2003), BITE maintains an organization hierarchy and a task/sub-task behavior graph to manage teamwork. However, in addition, BITE maintains a novel third structure, a library of hierarchically linked *social interaction behaviors* implementing interaction protocols for synchronization and task allocation. These are (re)used interchangeably, to automatically coordinate team-members' selection of task behaviors.

We describe our experience in using BITE with teams of Sony AIBO robots. BITE's structure gives rise to novel separation of social interaction from task-oriented control, and provides uniform access to interchangeable interaction protocols. We show in multiple experiments, that this separation and is a significant contribution, in the sense that it accounts for non-trivial effects on team performance, and facilitates robustness.

Motivation and Background

Our primary motivation is to explore architectural mechanisms for flexible teamwork. Teamwork literature reveals common teamwork primitives: sub-task *synchronization* (getting agents to temporally coordinate task execution), and task *allocation* (getting agents to divide up the subtasks between them). While previous work has addressed both of these important aspects of teamwork, it mostly focuses on one aspect at a time.

The TEAMCORE architecture (Pynadath & Tambe 2003) uses decision-theoretic communications in synchronizing the selection and termination of hierarchical behaviors, and uses task re-allocation behaviors that could be triggered

[*]Gal Kaminka is also affiliated with CMU. This research was supported in part by BSF grant #2002401, and by Israel's Ministry of Science infrastructure grant # 3-942.

based on catastrophic failures. The ALLIANCE behavior-based architecture (Parker 1998) focused on robustness, by allowing robots to dynamically re-allocate themselves to tasks, based on failures in themselves in their teammates. Both of these architectures have been demonstrated to work in multiple domains. TEAMCORE provides synchronization and some allocation services. ALLIANCE is offers dynamic task allocation, but does not explicitly synchronize robots as they jointly take on tasks. Both rely on fixed interaction protocols, and in that they are *monolithic*: They do not allow flexibility in choosing the interaction protocols underlying synchronization and allocation.

Within robotics work, a number of monolithic market-based approaches have been proposed for task allocation. Dias and Stentz (2000) discuss the use of markets to allow robots to bid for tasks in spatial sensing domain. Gerkey and Matarić (2004) explored multi-robot task allocation.

We believe that a teamwork architecture (1) should provide integrated synchronization and allocation; and (2) should be non-monolithic, in that different synchronization and allocation protocols could be mixed and matched, even within the same task. SCORE (Vu *et al.* 2003) demonstrated the usefulness of using multiple protocols depending on execution context. However, SCORE only allowed flexible allocation; its synchronization mechanism is communication-intensive and prone to failures.

BITE seeks to fulfill this vision of flexible teamwork in robot teams. It can provide many, though 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 interaction behaviors. Thus for instance, it is possible in BITE to switch between multiple synchronization methods, to dynamically re-allocate robots to tasks in more than one way, and to manage proactive communications. However, BITE still lacks STEAM and ALLIANCE's failure-recovery facilities.

BITE: Structures and Control

BITE uses hierarchical behaviors as the basis for its control. To these, it adds two additional structures: A set of social interaction behaviors, and an associated team-hierarchy. A single control algorithm uses these structures to automate control and communications of a team of robots

Control Structures

The first of the three structures specifies the sequential and hierarchical relationships between task-oriented behaviors. The *task behavior graph* is an augmented connected graph tuple $\langle B, S, V, b_0 \rangle$, where B is a set of task-achieving behaviors (as vertices), S, V sets of directed edges between behaviors ($S \cap V = \emptyset$), 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), 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 sequential edge from b_1 to b_2 specifies that b_1 must be executed before executing b_2. A path along sequential edges, i.e., a valid sequence of behaviors, is called an *execution chain*. 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 executes a complete path—root-to-leaf—through the behavior graph. Sequential edges may form circles, but vertical edges cannot. Thus behaviors can be repeated by choice, but cannot be their own ancestors.

Previous teamwork architectures rely on similar behavior graphs (with some variations) to represent and manage task control knowledge (Parker 1998; Pynadath & Tambe 2003). A common theme is for each robot to have its own copy of the behavior graph. Behaviors whose execution is to be coordinated in some fashion (henceforth, *team behaviors*) are tagged in advance by the designer. The teamwork architecture in question automatically take actions to select and de-select these in different robots, when appropriate.

Figure 1-b shows an example of a simple behavior graph, constructed for 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–*search*– in which robots visually search for their peers and their own relative locations. Then, 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. The above behaviors are tagged as team behaviors, and require two important teamwork capabilities: *synchronization* (to make sure all robots select the same behavior, and start/end *walk* or *linewalk* together), and *allocation* (to make sure only a single leader for the formation is chosen, the followers are assigned different relative positions, etc.).

To allow BITE to automate synchronization, we impose a constraint on the semantics of multiple outgoing edges. Two outgoing sequential edges $\langle a, b \rangle, \langle a, c \rangle$ 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. When these execution chains are composed of team behaviors, the selection between alternatives must be coordinated—all (relevant) robots must select the same execution chain (we discuss below complex cases in which only certain subteam members must coordinate). Thus BITE's synchronization (see below) is triggered when multiple execution chains are enabled, and the robots must coordinate their joint selection.

To automate allocation, we impose a related semantic constraint on decomposition edges. Two outgoing decomposition edges $\langle a, b \rangle, \langle a, c \rangle$ 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 point only to the first behaviors of execution chains). Thus such multiple outgoing edges indicate that the children (subtasks) can be allocated to different subteams. Therefore, similarly to the synchronization points, BITE's allocation services are triggered when multiple decomposition edges are enabled.

There is one final point in which synchronization is needed. Teamwork theory states that when an agent pri-

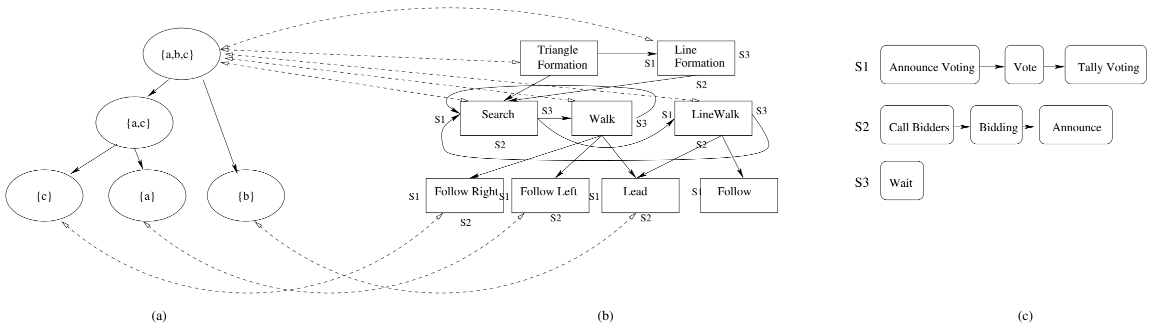

Figure 1: **BITE structures and links in a small example application**

vately believes that a joint goal has been achieved, or should be abandoned, the agent must make this belief mutual with its teammates. In practice, this corresponds to the robots terminating their execution of team behaviors in a coordinated fashion. Thus when a team behavior's termination conditions are satisfied for a robot, BITE is triggered to coordinate the termination of this behavior with the other robots.

To summarize, BITE can easily determine synchronization and allocation points given the constraints above. A split in sequence edges leading to team behaviors signifies a synchronization point. A split in decomposition edges leads to allocation. And synchronized termination is triggered when a team behavior is de-selected. In all of these, BITE must coordinate with the other robots (through their own BITE processes).

To carry out such coordination, BITE first needs to maintain knowledge about the robots that are responsible for coordinated execution of team behaviors. To do this, BITE maintains a second structure, the organization hierarchy (called the team hierarchy in (Pynadath & Tambe 2003; Vu *et al.* 2003)). 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 $\langle R_1, R_2 \rangle$, then $R_2 \subset R_1$.

To allow behaviors to reason about the organizational unit responsible for their execution (and vice-versa), we create bi-directional 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 (and back) if B_j is to be jointly executed by R_i. 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*. The robots must jointly decide how to allocate the different roles of the formation. One must lead the triangle (the *lead* behavior), while the others follow—from the left (*follow left*) and right (*follow right*). To negotiate this allocation, 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.

Previous architectures have made hard commitments to the protocols used for coordination. For instance, architectures that supported allocation utilized a market-based approach (Dias & Stentz 2000), or an agenda-based mechanism (Parker 1998).

A key novelty in BITE is that it allows the use of different interaction protocols at different times, depending on the team behaviors in question (and other context information). To achieve this, BITE maintains a third structure, holding a set of *social interaction behaviors* which control inter-agent interactions. Interaction behaviors typically control communications and execute protocols (e.g., voting) that govern coordinated activity. Each interaction behavior is encoded in a separate behavior graph. For instance, a simple synchronization behavior (by voting) may be decomposed into three atomic interaction behaviors, executed in sequence: Announce vote, tally votes, and announce winning selection.

In order to facilitate the execution of interaction behaviors, we link the task behaviors to interaction 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.

Synchronized selection is triggered 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 $S1$).

Allocation of sub-teams to behaviors is triggered 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 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 (Dias & Stentz 2000), or an agenda-based mechanism (Parker 1998). 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.

Finally, **synchronized termination of behavior execution** determines the social behavior of robots as they reach the end of an execution chain. Normally, upon such termination, 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 necessarily 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.

Social interaction behaviors may themselves require synchronization, and allocation. As interaction behaviors are represented using behavior graphs (as the task-oriented behaviors), they can themselves link synchronization, allocation, and termination points in their behavior graphs to other interaction behaviors, thus creating hierarchical social interactions. For instance, we have described a simple voting interaction behavior. To execute these, BITE may need to allocate the task of announcing the vote to one robot, have all robots synchronize the beginning and end of sending their votes, allocate someone to tally the votes, etc. Thus robots may end up using another synchronization behavior (e.g., one where the choices are pre-set by the designer), in order to execute another. However, as behavior graphs do not allow cyclic decomposition (and interaction behaviors use behavior graphs), an infinite cycle where robots vote as to how to vote, etc. is not possible in principle.

Principal Control Algorithm

Each of the robots executes Algorithm 1, using its own copy of the three structures. The control loop executes *a behavior stack*—root behavior to leaf—where top behaviors on the stack are executed simultaneously with their currently selected children.

Execution begins by pushing the initial behavior of the graph on the execution stack (lines 1–2). Then the algorithm loops over four phases in order. (i) It recursively expands the children of the behavior, allocating them to sub-teams if necessary (lines 3a–3c). (ii) 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 (iii) 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, (iv) 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.

Algorithm 1 CONTROL

```
Input: behavior graph $\langle B, S, V, b_0 \rangle$, team hierarchy $T$, interaction behaviors set $O$
1. $s_0 \leftarrow b_0$ // initial behavior for execution
2. push $s_0$ onto a new behavior stack $G$.
3. while $s_0$ is non-atomic // has children
   (a) $A \leftarrow \{b_i\}$, s.t., $\langle s_0, b_i \rangle$ is a decomposition edge
   (b) if $A$ has only one behavior $b$, $push(G, b)$.
   (c) else $b \leftarrow$Allocate$(G, s_0, A, T, O)$, $push(G, b)$.
   (d) $s_0 \leftarrow b$.
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 $Q \leftarrow \{s_i\}$, s.t. $\langle b_0, s_i \rangle$ is a sequential edge
   (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.
```

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 calls the appropriate social interaction behavior in O (linked from the current parent) to make the allocation decision. 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. In addition, the interaction behavior is given access to any links from the parent behavior to the team hierarchy, e.g., to determine whether any children task-behaviors are already pre-allocated. 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 interaction behavior, which can return a new child behavior for execution under the current parent. If it doesn't, then the next behavior in the execution chain must be selected by $Decide()$, which calls a synchronization interaction behavior. Since synchronized selection involves all members of the current sub-teams selecting together, this behavior would normally 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.

Additional algorithms can be derived based on analysis of the three structures and their interacting links. For instance, 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 (Veloso, Stone, & Bowling 1999). Indeed, we have found it useful to run a proactive communications algorithm that informs teammates of sensed knowledge that may be relevant to them (Algorithm 2—described in the next section). The team hierarchy and behavior graph provide the information as to whom should be informed of what.

BITE: Evaluation and Lessons Learned

The previous section has presented the key novel aspect in BITE—its introduction of social interactions as first-class objects, implemented as behaviors which are managed through the same mechanisms used to manage task-oriented behaviors. This feature of BITE allows for a significant degree of flexibility in teamwork over existing architectures. For instance, it allows creation of systems that marry the capabilities of different architectures. It also allows different interactions to be used depending on context.

This section reports on the lessons learned from fully implementing BITE and running it in a variety of tasks, on Sony AIBO robots. We first report on the benefits of BITE's flexibility—the key motivation for its development. We then report on additional features which were introduced into BITE as a response to the application of multi-agent systems technology in physical robots. While we are not the first to make the transition (see, for instance, (Dias & Stentz 2000) as examples of market-based allocation in robotics), we believe that the lessons learned from this application have not been previously reported.

Lesson: Flexibility Makes a Difference. The robots, running BITE, have been in used in several tasks. One specific task had the robots go from an initial position where they are all lined side by side to a goal location 6 meters away. The robots were to maintain a triangle or line formation (as described above—see also Figure 2). If moving in a triangle formation, the robots could choose to change to line based on some conditions. The behavior graph for this task is composed of a dozen task behaviors, and eight interaction behaviors. The task behavior graph has two allocation points and three synchronization points. While relatively simple, this behavior graph is sufficient to show that the ability to use a variety of interaction behaviors makes a significant difference in task performance.

In an initial simple experiment, we ran three separate versions of the tasks, where the same behavior graph (with both triangle and line formations) was used with different combinations of interaction behaviors. Each variant was run four times. We measured synchronization and allocation times, as well as overall task time. Table 1 shows the average results. The table summarizes the time spent in each type

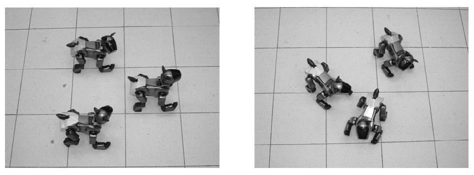
(a) Success (b) Failure

Figure 2: **Robots executing BITE.**

Policy	Sync Time	Alloc Time	Interaction Time	Task Time
SYNC.	2883.00	424.50	3307.50	56590.5
NO SYNC.	0	548.75	548.75	54683.0
SYNC/ALLOC.	2572.75	949.00	3521.75	45887.0

Table 1: **Interaction and task times (ms).**

of interaction, as well as the total interaction behavior time and the task completion time. In the first variant (SYNC), communication-intense interactions were selected for synchronization, but communication-poor interactions were selected for allocation. In the second (NO SYNC), both synchronization and allocation used as little communication as possible. In the third, both synchronization and allocation used communication-rich interactions.

Table 1 shows that flexibility matters—in non-trivial ways. Comparing the SYNC and NO SYNC variants, we see that a reduction in communications (by approximately 2 seconds) entailed a corresponding 2-second reduction in task time. This was to be trivially expected under the assumption that savings in interaction time translate into reduced task completion time. However, the third variant provides a surprise. Despite spending more time in interaction, it manages to reduce task completion time significantly (by about 19%). Note that total interaction time is almost the same as in the first variant—but now a much larger portion of it is spent in allocation interactions. This shows that the ability to mix and match interaction behaviors can be a significant factor in task performance, lending support to this novel feature being an important contribution.

Lesson: Automated Teamwork Saves. The formation behaviors operate by forcing robots to 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. This algorithm is inherently susceptible to sensing failures (see also another lesson below), as these cause the failing robot to lose its place in the formation (e.g., Figure 2-b). 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 loses track of the lead robot, the appropriate termination condition is satisfied, and the behavior terminates. This in turn triggers the appropriate social interaction behavior in BITE, which causes the other robots to also stop executing their movement behavior, and jointly switch to the

Graph #	Trials	Failures	Interaction Time	Task Time
1.	7	0	1831	52730
2.	5	1	1709	122801
3.	5	1	1284	54005
4.	5	0	2257	179464
5.	5	1	688	174563
6.	6	1	0	70251
7.	7	0	836	109240
8.	5	1	2900	120392
9.	7	1	572	105516

Table 2: **Results from 9 Behavior Graphs**

Search behavior. Thus the formation is maintained.

To show the savings generated by BITE, we created 9 different behavior graphs, by mixing a dozen individual and team formation-maintenance behaviors, and connecting them in various sequential orders, with and without optional cycles, etc. We assigned each of the 9 behavior graphs a set of social behaviors. Table 2 shows the results. The second column shows the number of successful trials; the next column shows the number of failed trials; the next two columns measure the average interaction and task completion times.

Failures consisted of the robots getting confused about their roles, e.g., because they found themselves on the same or wrong side of the leader in a triangle formation. It turns out that all failures could be traced to a specific non-communicating allocation behavior, which was faster (no waiting for messages), but less robust. Wherever a communicating allocation behavior was used, all such failures were prevented.

Other than these failures—which ultimately are due to a bad choice by the designer—BITE handled all nine behavior graphs with equal ease. There was no need to re-write communication rules, or to worry about how certain behaviors interact with each other. All transitions between coordinated modes took place automatically, as did all allocations, including all dynamic allocations of formation roles, based on positions of the robots after they have already executed some other formations.

Lesson: Fuse Sensor Information. Two mechanisms are used in BITE for sensor fusion. First, as common in robotics work, sensor data is pre-processed before being used. BITE also uses multi-robot sensor fusion, e.g. (Stroupe, Martin, & Balch 2001). However, it treats such procedures as interaction behaviors, which can be mixed and matched depending on the execution context.

Algorithm 2 shows a simple example fusion behavior. In the first phase of the algorithm, each robot determines whether new information affects its behavior stack (e.g., newly-satisfied conditions). These potentially affect the robot's teammates, and must therefore be communicated to them by finding out which sub-team is responsible for each behavior on the stack (done through the *Inform*() procedure, which refers to an appropriate social interaction behavior). The second phase (lines 2a–2b) allows the robot to determine whether newly sensed information may be relevant to sub-teams that it is not a member of, proactively providing them with information even though it is not strictly its own responsibility to do so.

Algorithm 2 FUSEINFORMATION

```
Input: behavior graph $\langle B, S, V, b_0 \rangle$, team hierarchy $T$, interaction behaviors set $O$
1. for all behaviors $b$ on behavior stack $G$:
   (a) $t \leftarrow subteam(b)$ // sub-team responsible for $b$
   (b) if a termination condition of $b$ is satisfied,
       $Inform(b, t, O)$
   (c) if a precondition of a behavior $f$ ($\langle b, f \rangle$ a sequence edge) is
       satisfied, $Inform(b, t, O)$
2. for all teams $t$ in the team hierarchy:
   (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, $Inform(b, t, O)$
       ii. if a precondition of a behavior $f$ ($\langle b, f \rangle$ a sequence edge) is
           satisfied, $Inform(b, t, O)$
```

Summary and Future Work

A key challenge in building robot teams is to automate teamwork, such that the designer can focus on the robots' taskwork. Existing teamwork architectures automate key aspects of teamwork, but do not allow flexibility in how this automation is achieved, in terms of the interaction protocols used. Nor do existing architectures allow for mixing different protocols within the same overall robot team mission.

We argue that teamwork should adapt a mini-kernel approach, allowing teamwork services to be interchanged and mixed as needed. Concretely, we presented BITE, a distributed behavior-based teamwork architecture that enables such flexibility 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 within this architecture, and the lessons generated in applying it in Sony AIBO robots. Empirical results from multiple experiments show that flexibility is indeed important, and can significantly affect task achievement. Our future efforts focus on human-team interactions, and on extending BITE's capabilities.

References

Dias, M. B., and Stentz, A. T. 2000. A free market architecture for distributed control of a multirobot system. In *6th International Conference on Intelligent Autonomous Systems (IAS-6)*, 115–122.

Gerkey, B. P., and Matarić, M. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research* 23(9):939–954.

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

Pynadath, D. V., and Tambe, M. 2003. Automated teamwork among heterogeneous software agents and humans. *JAAMAS* 7:71–100.

Stroupe, A. W.; Martin, M. C.; and Balch, T. R. 2001. Distributed sensor fusion for object position estimate by multi-robot systems. In *ICRA-01*, 1092–1098. IEEE Press.

Veloso, M.; Stone, P.; and Bowling, M. 1999. Anticipation: A key for collaboration in a team of agents. In *SPIE Sensor Fusion and Decentralized Control in Robotic Systems II (SPIE-99)*.

Vu, T. D.; Go, J.; Kaminka, G. A.; Veloso, M. M.; and Browning, B. 2003. MONAD: A flexible architecture for multi-agent control. In *AAMAS-03*.