Integration of Coordination Mechanisms in the BITE Multi-Robot Architecture

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

Abstract— **Recent years are seeing a renewed interest in general multi-robot architectures, capable of automating coordination. However, few architectures explore integration of multiple coordination mechanisms. Thus the question of how to best integrate coordination mechanisms is left open. This paper focuses on the *micro-kernel* integration approach used in BITE (Bar Ilan Teamwork Engine), a multi-robot behavior-based architecture. This approach allows the developer to plug in coordination mechanisms (teamwork behaviors) to be used depending on the context of execution. BITE imposes constraints on the specification of taskwork behaviors, which allow BITE's control algorithm to automatically determine sequencing and task-allocation points during task execution. At such points, teamwork behaviors (known as *interaction behaviors*) are triggered to automate the coordination processes. We argue that BITE's approach is preferable to the methodology of existing architectures, and provide analysis of experiments in using BITE with Sony AIBO robots, to support our argument.**

I. Introduction

Recent years are seeing a renewed interest in general (task-independent) architectures, capable of automating the coordination of multiple robots. Architectures such as ALLIANCE [12], TraderBot [2], Distributed 3T [7], [8], MURDOCH [5], BITE [10], and ASyMTRe-D [13], all provide the robots with automated coordination (*teamwork*), such that the developer can focus on designing the task-specific mechanisms, i.e., *taskwork*.

Existing architectures explore many facets of coordination and collaboration. Some focus on task allocation mechanisms (e.g., [2], [5], [8], [12]); others on distributed resource sharing and management (e.g., [7], [13]). However, with few exceptions (e.g., [7]), relatively few architectures explore integration of multiple coordination mechanisms. Thus the question of how to best integrate coordination mechanisms is left open.

This paper focuses on the *micro-kernel* integration approach in BITE (Bar Ilan Teamwork Engine), a multi-robot behavior-based architecture (see [10] for an early description). This approach allows the developer to plug in coordination mechanisms (teamwork behaviors) to be used depending on the context of execution. BITE imposes constraints on the specification of taskwork behaviors, which allow BITE's control algorithm to automatically determine sequencing and task-allocation points during task execution. At such points, teamwork behaviors (known as *interaction behaviors*) are triggered to automate the coordination processes.

BITE's micro-kernel approach differs from existing architectures, that are *monolithic*, in the sense that their coordination mechanisms are essentially hard-wired into them, and form part of the architectures. Earlier work has provided initial evidence that flexibility in selecting coordination mechanisms (depending on the context) can lead to improved performance, even when the task-related skills themselves do not change [3], [10]. Thus we argue that BITE's approach (which allows such flexibility) is preferable. This paper explores this argument in depth.

BITE has been implemented for use with the Sony AIBO robots, and for robots supporting the player-stage API [6]. We provide an in-depth discussion of BITE's structure and control mechanism, and detailed results from a number of experiments using BITE with Sony AIBO robots. These experiments demonstrate key capabilities in BITE. We additionally explicitly note BITE's weaknesses, in the hope that these clarify important open issues.

This paper is organized as follows. Section II discusses related work. Section III discusses the structure of BITE, and how it facilitates integration coordination mechanisms. Section IV presents qualitative and quantitative evaluation of BITE's approach. Section V concludes.

II. Background and Motivation

Interest in architectures for coordinating multiple cooperating robots has generated vast amounts of relevant literature. We therefore focus on the most related efforts in this area, specifically restricting ourselves to task-general architectures.

Our primary motivation is to explore architectural mechanisms for flexible teamwork. Teamwork literature reveals common teamwork primitives: Sequential task *synchronization* (getting agents to temporally coordinate task execution), and task *allocation* (getting agents to divide up subtasks between them). Our initial motivation in BITE was to marry both of these aspects in a single architecture.

One general difference between BITE and all of the following architectures is that they rely on fixed interaction protocols, and in that they are *monolithic*: They do not allow flexibility in choosing the interaction protocols to suit task context, but rather utilize fixed built-in protocols.

*This was was supported in part by BSF Grant #2002401.

Moreover, most architectures focus only on specific aspects of teamwork (e.g., task allocation), and do not include other mechanisms (e.g., synchronized task sequencing). More specific differences are discussed below.

One area of related work comes from multi-agent systems literature. BITE follows the footsteps of STEAM [16] and TEAMCORE [17] in that it is behavior-based, and in that it shares with them the use of a team-hierarchy to keep track of team-members task assignments. However, in addition to the differences noted above, these earlier works have not been applied to controlling multiple robots, and do not include the world-modeling components which BITE includes for this purpose.

The ALLIANCE behavior-based architecture [12] focuses on robustness, by allowing robots to dynamically re-allocate themselves to tasks, based on failures in themselves in their teammates. ALLIANCE offers dynamic task allocation, but does not explicitly synchronize robots as they jointly take on tasks. However, it offers robustness "out-of-the-box", while robustness in BITE depends on the choice of interaction behavior.

There has been considerable interest recently in using auction and market-based task-allocation methods in multi-robot systems. TraderBot [2] explored the use of markets to allow robots to bid for tasks in spatial sensing domains. Goldberg et al. [8] explore a distributed three-tier architecture, in which multiple robots interact with each other at all three layers using a market-based resource allocation scheme. Gerkey and Matarić [5] discuss the use of such methods in contrast to others. BITE complements such work by integrating task allocation with synchronized sequencing of tasks. It also offers the potential for using multiple (different) allocation methods within the same application, depending on the execution context.

Farinelli et al. [4] explore alternative novel methods for task allocation in robot teams. Their token-passing method are suitable for teams of larger scale than BITE is typically deployed with. However, it should be possible in principle to build a token-passing allocation mechanism for BITE as well.

Jung and Zelinsky [9] have explored the use of a distributed architecture, which—like BITE—is behavior-based. However, the focus of this work is on cooperative spatial planning, rather than integrating coordination mechanisms. Similarly, Alur et al. [1] offer a comprehensive framework for spatial coordination of multiple robots. In contrast, BITE has no planning capabilities, and is not specific to spatial coordination. Rather, spatial coordination mechanisms can be (and have been) integrated using BITE's integration approach.

Goldberg et al. [7], [8] have explored a different basis for multi-robot architectures. While BITE takes a behavior-based approach to control, Goldberg et al. focus on extending a 3-tier architecture with an impressive set of capabilities, including task-sequencing and task-allocation, and distributed resource management. BITE offers the potential to utilize a wider spectrum of protocols (since none are built in), but currently lacks some of these capabilities.

ASyMTRe-D [13] has been recently developed to explore sophisticated mechanisms for distributed resource management and resource sharing. In contrast, BITE does not (yet) allow for resource sharing. However, it integrates coordination mechanisms that are lacking in ASyMTRE-D.

BITE's integration approach separates the coordination mechanisms (known also as interaction protocols) from the architecture. It runs a *behavior management kernel* which manages resources, selects behaviors for execution, and terminates their execution when needed. All coordination mechanisms lie outside of this manager, in a separate library of interaction behaviors, which can be used in a mixed fashion, from different points during task execution.

The ability to call on interaction behavior in a execution context is motivated by our own early work on SCORE [3], which demonstrated the usefulness of using multiple protocols depending on execution context. However, SCORE only allows flexible allocation; its synchronization mechanism is communication-intensive and prone to failures.

An earlier version of BITE has been previously described [10]. However, compared to the earlier work, this paper reports on important revisions to the architecture, and new results that have not been previously published.

III. The Structure of BITE

BITE is divided into two computational components. The first component is a *world-modeling process*, which is responsible for processing sensor and communication data, and for sharing this information (when needed) with other team-members. The second component is the *control process*, which runs the main behavior-manager algorithm (selecting and deselecting behaviors). These two components are described below. For the purpose of clarity, however, we begin by describing the control process (Section III-A) and only then describe the world-modeling process (Section III-B).

A. Control Process

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.

1) 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 and V are sets of directed edges between behaviors ($S \cap V = \emptyset$), and $b_0 \in B$ is 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.

As with existing architectures (e.g., [12], [14]), each robot executes a local 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 takes 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 privately believes that a joint goal has been achieved, or should be abandoned, the agent must make this belief mutual with its teammates. The communication of beliefs is handled by the collaborative world modeling process (next section). Here, the implication is that robots must terminate 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 [3], [14]). 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 $R_2 \subset R_1$. 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 organizational unit responsible for their execution (and vice-versa), we create bi-directional links between the behavior graph and the team hierarchy. A link from a behavior B_j points to a sub-team R_i (and back) if B_j is to be jointly executed by R_i.

To allow reasoning about allocated tasks, we maintain the inverse links as well, from sub-teams to the behaviors they are responsible for. For instance, in Figure 1-a, one can see a team hierarchy composed of three robots, simply identified as a, b, 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, executing the formation task (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

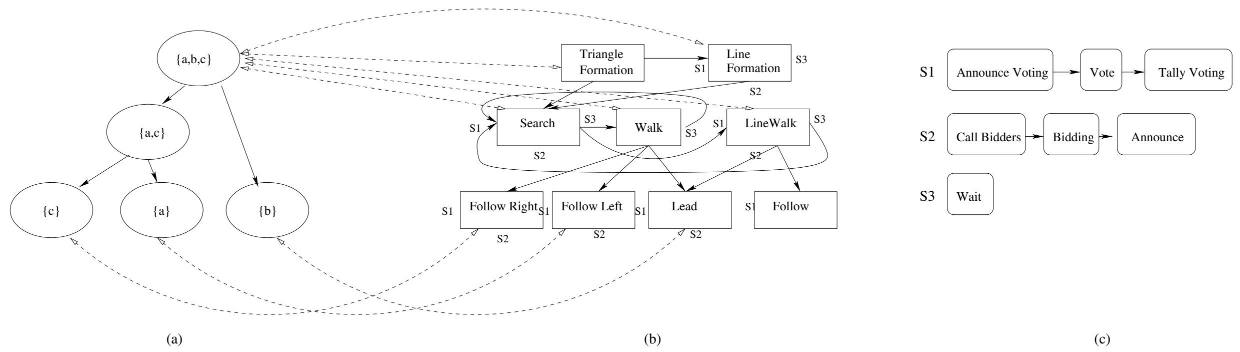

Fig. 1. **BITE structures and links in a small example formation-following task.**

multiple decomposition edges signify an allocation choice. 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.

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 $S1$). Here, 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 is triggered when a behavior is to be decomposed into children behaviors. If only one decomposition transition exists, then the team selects it. Otherwise, if multiple decomposition transitions exist, then the team is split into sub-teams. The appropriate social behavior is called to carry out this allocation, for instance by using a market-based approach [2], [8]. 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.

Recursive social interactions. 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 it, 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.

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.
```

2) 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, and then a final goto (line 9) back to line 3 begins again.

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, e.g., 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. We leave such analysis for future work.

B. Collaborative World Modeling

Recent versions of BITE introduced the world-model as a separate computational process. This process carries out both traditional sensor-filtering and processing, it also executes collaborative algorithms, which share information with other robots. We focus on this aspect here.

While social interaction behaviors may exchange information as needed for the protocols they implement (e.g., votes), there are more basic communication needs that underlie behavior execution. To illustrate, consider the following example: Suppose a robot has determined that a running behavior is to be terminated (because its termination condition matches). A call is made to a synchronized-termination social behavior. But as it contacts the other robots, it refers to information that is known only to the robot initiating the dialog. Obviously, the termination protocol can be fixed such that it first transmits the missing information, and then argues for termination. But rather than duplicate this functionality in all termination protocols, it makes sense to allow this to happen—in a flexible manner—in the world-modeling process.

Indeed, the world-modeling process can execute distributed information-sharing algorithms. The algorithms can be as simple as an algorithm that updates all team-members with any change in perception; or it could be complex and sophisticated, able to consider uncertainties in fusing information about the world [15].

Algorithm 2 FuseInformationWithTeammates

```
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*$(b, t, O)$
```

Algorithm 3 ProvideHelpfulInformation

```
Input: behavior graph $\langle B, S, V, b_0 \rangle$, team hierarchy $T$, interaction behaviors set $O$
1) 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*$(b, t, O)$
```

Through BITE's lifetime, we have experimented with different approaches here. The simplest one broadcasts all changed information. For obvious reasons of bandwidth usage, it was quickly ruled out in favor of a more focused algorithm, which broadcasts information about changes in the preconditions and termination conditions of currently-executing behaviors of team-members. This agrees with theoretical notions of teamwork, which specify that team-members that privately come to belief a proposition relevant to the team, must establish mutual belief in this proposition [11]. Thus such information is only broadcasted to relevant team-members (i.e., to team-members currently executing a behavior affected by the new information). The algorithm appears below (Algorithm 2).

In Algorithm 2, 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).

A second algorithm which we find useful addresses helpful communications, in which a robot communicates relevant information even when it is not strictly its responsibility, i.e., in the case where the information is relevant to a subteam other than its own. This algorithm (Algorithm 3) uses the team-hierarchy to discover which subteam is responsible for a specific behavior, so that relevant information can be communicated to it.

Algorithm 3 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. The two algorithms may be run with different priorities within the collaborative world-model. This will allow obligatory messages for teammates to receive higher priority than messages that are not required.

For instance, suppose a team of robots is executing the formation task described above, including the sub-team allocations (Figure 1). Suppose that the robots are executing the *Triangle Formation* behavior, and below it, the *Walk* behavior. One robot is executing the *Follow Left* behavior. 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. Algorithm 3 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.

IV. Evaluating BITE

Evaluating an architecture such as BITE is notoriously difficult, since there are multiple aspects to such evaluation. We consider key aspects one by one below.

A. What BITE Can Do

BITE is fully implemented and is used on both Sony AIBO robots, as well as robots working with the player-stage API [6]. It is used by students in our research group to facilitate coordination in multi-robot tasks such as coverage and formation maintenance.

One aspect of evaluating BITE is in establishing whether the ability to use multiple coordination mechanisms matters. For instance, can robot teams really benefit from using different task-allocation mechanisms at different times?

Preliminary results [10] have provided initial evidence that (1) BITE is flexible (in the sense of being able to use multiple coordination mechanisms in the same task); (2) such flexibility matters (in that selection of different coordination mechanisms affects performance in non-trivial ways). Here, we analyze this flexibility in more depth.

We created a behavior-graph composed of hierarchical task-behaviors for moving Sony AIBO robots in triangle and column formations. The behavior graph is shown in Figure 2-a. The numbers on each node indicate the node identification (used in the discussion below). The points a–c mark synchronization points; d, e mark allocation points.

We then defined several different social interaction behaviors, described in Table I. The table shows the name of each behavior (typically, beginning with SF), its type (synchronization or allocation), and whether the behavior uses communications. The final column provides a short description of the protocol implemented by the interaction behavior. For instance, whether the decision is based on the robot's ID (known to all), or by a voting mechanism (which we will not describe here for lack of space).

Using these behaviors, we created several different configurations for execution by the AIBO robots. These configurations are shown in Table II. All of these configurations went through the behavior graph in order (behaviors 0-1-5-9-13-17-21). Each of these configuration were run multiple times (the number of runs appears in the last column). In all runs,

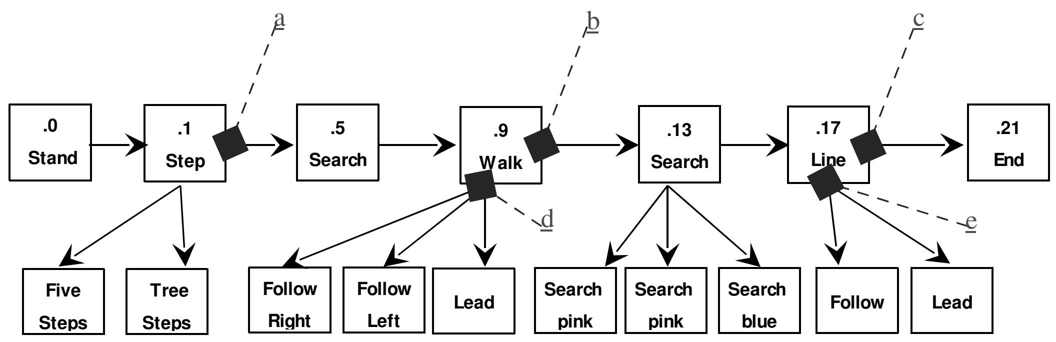

Fig. 2. A fixed behavior graph used in experiments with BITE.

SB Name	Transition Type	Comm.?	Description
SF1	Synchronization	No	Select based on ID
SF2	Allocate	No	Assign based on ID
SF3	Allocate	Yes	Select based on head angle
SF4	Synchronization	Yes	Formation leader decides on the next behavior and informs all its members (the decision is made based on number of times this behavior is executed)
SF5	Synchronization	Yes	Team members decide on a leader. After execution of the behavior, the chosen leader (randomly) decides on the next task to be executed.
SF6	Allocate	Yes	Assign based on the robot's color
SF7	Synchronization	No	Robots decide on the next behavior based on the number of times this behavior is executed
SF8	Synchronization	Yes	Voting
SF9	Synchronization	Yes	Complex voting.

TABLE I

SOCIAL INTERACTION BEHAVIORS DEFINED IN THE EXPERIMENTS.

Configuration	Executed social behavior point A-B-C-D-E	# Runs
1	SF1-SF1-SF2-SF1-SF2	6
2	SF1-SF9-SF2-SF1-SF2	5
3	SF1-SF9-SF2-SF1-SF2	7
4	SF5-SF1-SF2-SF1-SF2	5
5	SF8-SF9-SF3-SF1-SF2	7
6	SF1-SF7-SF3-SF4-SF6	5
7	SF5-SF1-SF2-SF5-SF2	5

TABLE II

BEHAVIOR-GRAPH AND SOCIAL BEHAVIOR CONFIGURATIONS.

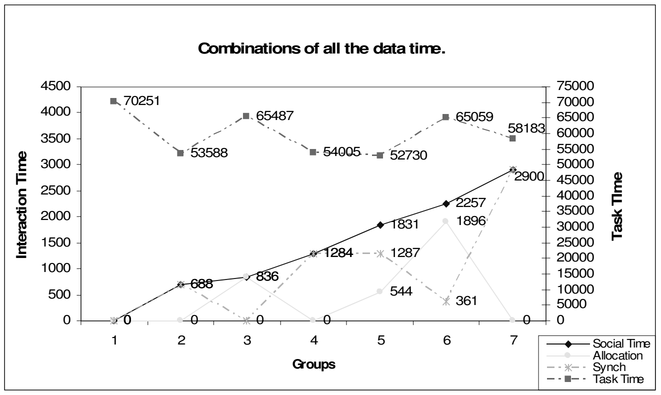

Fig. 3. Results for 7 different configurations of the behavior graph in 2, with different social behaviors.

the robots used the behavior graph to move in formation, in a straight line, for a fixed distance of 6 meters. The team-members were responsible for maintaining the formation, making coordinated decision about switching from triangle to column formations.

The results of the different runs appear in Figure 3. The right Y axis measures total task execution time (for movement of a fixed distance). The left Y axis measures the time spent in interaction, i.e., in social interaction behaviors. The line marked *task time* shows the average time to complete the 6 meter course in each configuration (averaged across all trials). The other lines measure the total interaction time, and its constituent factors: Time spent in synchronization and time spent in allocation.

Figure 3 shows that there are significant differences (almost 20 seconds) between the total task execution time of the best and worst configurations. Remember here the behavior graph is fixed. Thus the only cause for change is the selection of different configurations of coordination mechanisms (interaction behaviors). Moreover, one can see that in fact it is difficult to find correlation between the total interaction time and task performance. Thus we can rule out simple explanations that attempt to draw simple conclusions that more communications improve performance.

The implication is clear: The best-performing configuration is very specific, in the sense that it uses a very specific combination of synchronization and allocation behaviors. In this case, for instance, the choice of SF9 for a synchronization behavior in point b appears to make one key difference. But it does not account for the result by itself.

A second important evaluation criteria is demonstrated by these experiments. In running these experiments (and others), BITE did not require any programming effort for coordination or communications. Indeed, all configurations were easily executed by BITE. This may seem like a trivial result, in the sense that this is what we designed BITE to do. But this should be taken in contrast to the state of the art in multi-robot systems, where we often tweak coordination and communication rules in very specific places, to account for unforeseen circumstances. BITE's principled design allows such interventions when needed (in the form of new social interaction behaviors), but is focused on removing the need for them. Since all information sharing is done automatically, and all decision making is automatic, there is very little (if any) need for ad-hoc coordination.

B. What BITE Cannot Do

We believe BITE's micro-kernel approach to integration is superior to the monolithic architectures currently in use. Quite simply, there is no way to plug-n-play different coordination methods into an architecture, without introducing a kernel-like mechanism responsible for executing them.

Nevertheless, BITE is not perfect. BITE's design required making some choices that necessitate compromises on several issues. This subsection explicitly points them out, and we trust the reader to put them in the context: These are not negative results with BITE, simply a part of the evaluation examining the scope of BITE's capabilities.

BITE's commitment is to carry out task allocation only in the context of an agreed-upon top-level task (whose execution and termination is synchronized). This is fine for strongly-coordinated teams, but may be problematic for loosely-coupled coordinated groups, where we do not necessarily want robots to have to get all robots' agreement on the assigned task. Indeed, it has been shown repeatedly that a dose of aggression and competition is healthy even in cooperative settings (e.g., [18]). It is possible to model adversarial or competitive allocation protocols in BITE, but only once agents have agreed to the competition.

BITE's strong commitment to agreement can also hinder reactive responses. Say a robot is busy executing a task assigned to it in agreement with the team. If an emergency occurs that preempts this task, BITE will cause it to execute a synchronized-termination interaction behavior, which may or may not be sufficiently quick to execute (varying from doing nothing, to elaborate repeated acknowledgments and augmentations). This means that the robot's ability to respond to the emergency depends on the protocol chosen by the designer, rather than on the robot's knowledge (which in this case is correct, and causes it to want to switch).

V. Conclusions and Future Work

We have presented BITE, a behavior-based multi-robot architecture, which takes a unique approach to integrating together coordination and teamwork mechanisms. BITE separates these mechanisms from the architecture, and allows the developer to use (and build) a library of interaction behaviors which implement coordination protocols. To do this, BITE's task description structures impose structural and semantic constraints, which allow BITE to automatically determine when and how to execute coordination behaviors. We have shown that BITE, fully implemented for real robots, leads to being able to easily try different configurations of task behaviors and coordination mechanisms. We have also discussed BITE's limitations. Future work includes expanding BITE's capabilities to address distributed resource management and improved collaborative world-modeling.

Acknowledgments. We thank Ari Yakir, Tom Shpigelman, and Dan Erusalimchik for very useful discussions. As always, thanks to K. Ushi.

References

[1] R. Alur, A. Das, J. Esposito, R. Fierro, G. Grudic, Y. Hur, R. V. Kumar, I. Lee, J. Ostrowski, G. J. Pappas, B. Southall, J. Spletzer, , and C. J. Taylor. A framework and architecture for multirobot coordination. *International Journal of Robotics Research*, 21(10–11):977–995, October-November 2002.

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

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

[4] A. Farinelli, L. Iocchi, D. Nardi, and V. A. Ziparo. Assignment of dynamically perceived tasks by token passing in multi-robot systems. *Proceedings of the IEEE*, 2006. Special issue on Multi-Robot Systems.

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

[6] B. P. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics*, pages 317–323, 2003.

[7] D. Goldberg, V. Cicirello, M. B. Dias, R. Simmons, S. Smith, and A. T. Stentz. A distributed layered architecture for mobile robot coordination: Application to space exploration. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, 2002.

[8] D. Goldberg, V. Cicirello, M. B. Dias, R. Simmons, S. Smith, and A. T. 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.

[9] D. Jung and A. Zelinsky. An architecture for distributed cooperative planning in a behaviour-based multi-robot system. *Robotics and Autonomous Systems (RA&S)*, 26:149–174, 1999.

[10] G. A. Kaminka and I. Frenkel. Flexible teamwork in behavior-based robots. In *AAAI-05*, 2005.

[11] H. J. Levesque, P. R. Cohen, and J. H. T. Nunes. On acting together. In *AAAI-90*, Menlo-Park, CA, 1990. AAAI Press.

[12] L. E. Parker. ALLIANCE: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

[13] L. E. Parker and F. Tang. Building multi-robot coalitions through automated task solution synthesis. *Proceedings of the IEEE*, 94(7):1289–1305, 2006. Special issue on Multi-Robot Systems.

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

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

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

[17] M. Tambe, D. V. Pynadath, N. Chauvat, A. Das, and G. A. Kaminka. Adaptive agent integration architectures for heterogeneous team members. In *ICMAS-00*, pages 301–308, Boston, MA, 2000.

[18] R. Vaughan, K. Støy, G. Sukhatme, and M. Matarić. Go ahead, make my day: robot conflict resolution by aggressive competition. In *Proceedings of the 6th int. conf. on the Simulation of Adaptive Behavior*, Paris, France, 2000.