Bar-Ilan University
Department of Computer Science

Flexible Teamwork in Behavior-Based Robots, BITE (*Bar Ilan Teamwork Engine*).

by

Inna Frenkel

Submitted in partial fulfillment of the requirements for the Master's degree in the department of Computer Science

Ramat-Gan, Israel
2005

This work was carried out under the supervision of

Dr. Gal A. Kaminka

Department of Computer Science, Bar-Ilan University.

Abstract

The key challenge of deploying teams of robots in real world applications is to automate the control of teamwork, such that the designer can focus on the implementation of the tasks that the robots have to perform. Existing teamwork architectures that seek to address this challenge are monolithic, such that they are committed to interaction protocols at the architectural level and do not allow the designer to choose the most appropriate protocols for a given task. Moreover, most of the teamwork architectures have not been applied to physical robot teams. Therefore, they lack a number of key architectural features that are important for robots in physical environments. We present a behavior-based teamwork BITE architecture that automates collaboration of physical robots in a distributed fashion. BITE separates task behaviors that control the interaction of the robot with its task from interaction behaviors that control the interaction of the robot with its teammates. This distinction provides flexibility and modularity in terms of the interactions used to make collaboration between teammates effective. It also allows BITE to synthesize and significantly extend existing teamwork architectures. BITE also incorporates the conclusions drawn from earlier research by 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.

Acknowledgments

I would like to express my gratitude to:
Dr. Gal Kaminka - for his support and understanding during the course of research, for his attentiveness and interest.

For Kat'a Tryshkin, Maarten Beek and Natali Kanevsky, who helped all the way by commenting and reading this thesis.

Mr. Maor Asulin - for the assistance in creation of baseline test environment and simulation. He evaluated all the possible solutions, when one of the robots malfunctioned and the research was to be stopped, and gave this research a chance to be preceded.

Mr. Tom Shpigelman - for his professional skills and knowledge at creation of films, presentations, etc.

Mr. Danny Simony - for assistance at filming the experiment.

Ms. Ruti Glick and Mr.Yehuda Elmaliach - for the encouragement.

Contents

Abstract.....3

Acknowledgments.....4

Introduction.....9

Motivation.....9

Proposed.....10

Thesis Outline.....11

Background.....13

Methodology.....16

Representation.....16

Algorithms.....25

Principal Control Algorithm.....25

Resource Precondition Control Algorithm.....29

Fuse Information.....29

Operation Control Algorithm.....31

A Simple Execution Example.....33

Hardware Configuration and Software Applications.....41

The AIBO.....42

Tools.....43

OPEN-R.....43

Tekkotsu.....44

Project's Choice.....45

BITE.....46

BITE Installation.....47

BITE: Experiments*.....*49

Experiments Setup.....49

Experiments Data.....52

Results and Conclusions.....54

Errors.....61

Operator Control.....64

Summary and Future Work*.....*65

Bibliography*.....*67

List of Figures

Figure 1. Behavior graph.....17
Figure 2. Organization hierarchy.....20
Figure 3 BITE Structures and Links for a formation maintenance task.....23
Figure 4 Triangle Formations.....35
Figure 5. Executed *StandUp* behavior.....36
Figure 6. State of the team.....38
Figure 7. Position of each robot in Triangle Formations.....40
Figure 8. Termination of *Walk* behavior execution by all the robots.....41
Figure 9. Aibo front and back features.....42
Figure 10. Layer diagram.....47
Figure 11. Behavior graph.....50
Figure 12. Robots executing triangle formation.....63

List of Tables

Table 1. Social behavior profiles used for the experiments.....52
Table 2. Experiment group type is based on which behavior nodes in the graph were traversed and which social behaviors were executed by robots in the experiment.....53
Table 3. Time measurements for executing social behaviors.....55
Table 4. Average time spent on allocation and synchronization and total average time of the experiments for each group.....56
Table 5. Interaction and task times 61

List of Graphs

Graph 1 . Relation between the social time and task time.....56
Graph 2. Allocation and synchronization needs for each group.....57
Graph 3. Combination of the all data time.....58
Graph 4. Relation between number of error experiment and total experiment.....63

Chapter 1

Introduction

1.1 Motivation

Teamwork of autonomous robots gains more and more interest among academic and industrial research groups, motivated by the deployment of autonomous and semi-autonomous multi-robot teams in real-world applications. Teamwork architectures are increasingly used for the implementation of automated interactions between team-members to facilitate robust and speedy deployment.

Existing architectures provide important teamwork services to deployed teams, such as synchronized task execution [5, 9], and task allocation [6, 1, 3, 10]. These allow the designer to focus on developing the work to be done, rather than on the collaboration of the robots.

However, existing architectures leave important challenges open when applied to multi-robot teams. *First*, most existing robot teamwork architectures do not address synchronized task execution and dynamic task allocation in a single architecture. Therefore, the team's developer must decide whether synchronization or allocation is more important. *Second*, most of the existing architectures are monolithic in the sense that they commit only to specific services and to the specific interaction protocol used to implement each service, e.g., confirm-request for synchronization [9]. *Finally*, while previous work in robotics began to explore the use of task-allocation in physical robot teams [6, 1, 3], the actual application of multi-agent techniques to robotics has not been researched yet.

This thesis describes the Bar-Ilan Teamwork Engine (BITE), a novel behavior-based teamwork architecture targeting physical robot applications and addressing the challenges mentioned above. BITE integrates and synthesizes many features of existing teamwork architectures, and offers novel features. The architecture of behaviors is very flexible and comfortable to work with and it allows a designer to easily change and adjust the architecture to re-define the mission for the multi-robot teams.

BITE can be used for testing algorithms that are designed for multi-robot teams, for games that employ multiple robots such as GameBots or RoboCup, and even for synchronizing tasks of robot teams at an assembly line.

1.2 Proposed BITE System

We propose a novel behavior-based teamwork architecture targeting physical robot applications such as Sony AIBO platforms. This work represents our first step towards a comprehensive framework of controlling teams of behavior-based agents and robots, synthesizing and integrating many existing algorithms, and adding novel capabilities.

Similarly to previous architectures [9, 10], BITE maintains an organization hierarchy and a task/sub-task behavior graph of managing teamwork. However, previous architectures are monolithic and have utilized only a task/sub-task behavior hierarchy, or a behavior hierarchy with organization hierarchy. Next to these two hierarchies, BITE maintains a novel third structure, a set of hierarchically-linked social interaction behaviors implementing interaction protocols for synchronization and task allocation. These social behaviors handle interactions between the robots, and are triggered upon successful or failed termination of the individual behaviors of robots.

Thus, BITE architecture framework relies on maintaining and linking three graph structures:

- An organization hierarchy,
- A task/sub-task behavior hierarchy,
- A set of social behaviors.

The key idea of employing these three components is to simplify team control algorithms by separating control of individual behavior execution from the control of team-members interactions. Our proposed architecture implements a teamwork micro-kernel approach, in which different synchronization and allocation protocols can be used interchangeably and mixed as needed.

BITE integrates and synthesizes many features of existing teamwork architectures, and offers novel features. In addition, BITE incorporates features that stem from our research by applying multi-agent system techniques in physical robots. In particular, BITE emphasizes preprocessing of sensory information before use (including fusion of information from multiple robots) and facilitates human operator control.

We performed our experiment involving the BITE architecture on teams of Sony AIBO platforms. The software components utilize and extend the Tekkotsu AIBO control software [11]. In multiple experiments, we show that BITE's novel separation of social interaction from task-oriented control is an empirically significant contribution. This way, non-trivial effects on team performance can be accounted for. We additionally discuss conclusions drawn from applying multi-agent system techniques to robots.

1.3 Thesis Outline

From a design point of view, the BITE architecture is a large multiple-component-based project that was designed, developed and integrated using an object-oriented methodology. The project had been implemented through several major stages in approximately

two years. At each stage, a major system component was implemented and tested through laboratory experimental setups. The test results were analyzed and documented.

This thesis describes how the system was built from individual components. Each chapter of this thesis describes one of the components. The thesis comprises of eight chapters:

- **Chapter 2 *Background***

  Related work in multi-agent teamwork architecture is summarized here and compared to the proposed BITE system.

- **Chapter 3 Methodology**

  This chapter provides details of the design and algorithms of the BITE architecture. It also presents a simple example of the BITE execution to demonstrate how the main components of the architecture are related

- **Chapter 4 Experiments**

  This chapter describes implementation and results, conclude the thesis.

- **Chapter 5 Summary and Future Work**

  This chapters outline the future work for possible improvements and upgrades that would allow robots to automatically find and choose the most suitable mechanism of the team control and the negotiation protocol depending on the task settings.

The main contributions of this thesis was presented at the National Conference on Artificial Intelligence (AAAI-2005, 2005) [16] and at the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-05, 2005) [17]. Significant portion of this work has been presented at the Eighth Conference on Intelligent Autonomous Systems (IAS-8, 2004) [18].

Chapter 2

Background

Our work seeks to examine the application of multi-agent teamwork techniques to robotic teamwork and is inspired by research that has tackled teamwork at an architectural level. The investigation of existing teamwork literature has revealed parallel research lines. They focus on sub-task *synchronization* (coordination of robots task execution) or on task *allocation* (allocating subtasks to robots) one at a time. Thus our initial motivation is to combine both aspects of synchronization and allocation into a single architecture.

A second important motivation for this work has been to investigate the boundaries of existing techniques of multi-agent systems in physical robot applications. While there have been several successful applications of such techniques within robotics (e.g., [3, 4, 1]), there has been little done to investigate the characteristics or properties analysis of these techniques.

Tambe's work on the STEAM teamwork model [12] has explored the use of automated decision-theoretic communications in synchronizing the selection and termination of hierarchical behaviors. Additionally, it investigated the use of organizational repair behaviors that could be triggered based on failures in the organizational structure (such as a teammate permanently crashing). Our current work on BITE does not, as of yet, contain such monitoring and repair procedures.

Previous architectures have made hard commitments to the protocols used for coordination. For instance, architectures that supported allocation utilized a market-based approach [1, 4], or an agenda-based mechanism [6]. Other architectures utilized specific

protocols for synchronization. TEAMCORE [9] is an architecture that uses automated decision-theoretic communications and applies a confirm-request protocol in synchronizing selection and termination of hierarchical behaviors. I it also uses task re-allocation behaviors that could be triggered based on catastrophic failures. TEAMCORE provides synchronization and some allocation services.

Yen et al.'s CAST architecture [13] includes a mechanism for proactive communications, in which virtual robots anticipate the information needs of their teammates and respond appropriately. However, all synchronization is done via a simple fixed protocol, and allocation services are not provided.

SCORE [10] demonstrated the usefulness of applying multiple allocation protocols depending on execution context; however, SCORE only allows flexible allocation. Its synchronization mechanism is based on synchronizing a table of pre-declared variables and assumes that the latest value (communicated by anyone) is the correct one for all other robots. This protocol is communication intensive and prone to failures (for instance, it assumes robots always sense compatible information).

Within robotics work, Parker's work on the ALLIANCE architecture [6] has investigated robust distributed behavior-based control architecture, in which robots dynamically allocate and re-allocate themselves to tasks by sensing their own failures and those of their teammates. All ALLIANCE architectures have been demonstrated to work in multiple domains. They offer dynamic task allocation facilities, but do not explicitly synchronize robots that are jointly performing tasks.

Researches in the field of robotics have investigated a number of market-based approaches for robot task allocation. However, these are monolithic in nature and commit only to specific protocols. Dias and Stentz [1] discuss the use of markets to allow robots to bid for tasks in spatial sensing domain. Gerkey and Mataric [3] explore actions for task

allocation challenges, and also analyze several task-allocation protocols useful in robotics [14]. SCORE [10] 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 performing tasks jointly. CAST provides proactive communications, but does not provide many of the services in STEAM.

Additional recent work in multi-robot systems underscores the need for a comprehensive framework that facilitates automation of teamwork services. Goldberg et al. [4] explore a distributed architecture based on the traditional three-tier architecture, in which multiple robots interact with each other at all three layers using a market-based resource allocation scheme. 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 use other methods.

To rapidly deploy robotic teams, we believe that teamwork architecture must integrate many of the above mentioned capabilities and thus synthesize novel features. 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 of flexible teamwork in robot teams. Our work still lacks the failure-recovery facilities of STEAM or ALLIANCE; however, it provides many other capabilities of previous research, and in addition it separates 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.

Chapter 3

Methodology

Given the popularity of behavior-based control in existing robotics work, BITE uses hierarchical behaviors as the basis for the representation of underlying the controllers for the team members. In addition it has a set of social interaction behaviors with associated team-hierarchy which are described in *Section 3.1. Section 3.2* presents a number of principal algorithms that automate control and communication of a team of robots. *Sections 3.3* and *3.4* describe the experimental setup and give an example for a simple execution of the BITE.

3.1. Representation

SThe BITE system has three fundamental structures: *behavior hierarchy*, *organization hierarchy* and *social interaction behaviors*. They are designed to break down the monolithic structure of existing architectures.

Behavior hierarchy

Behavior hierarchy specifies the sequential and hierarchical relationships between

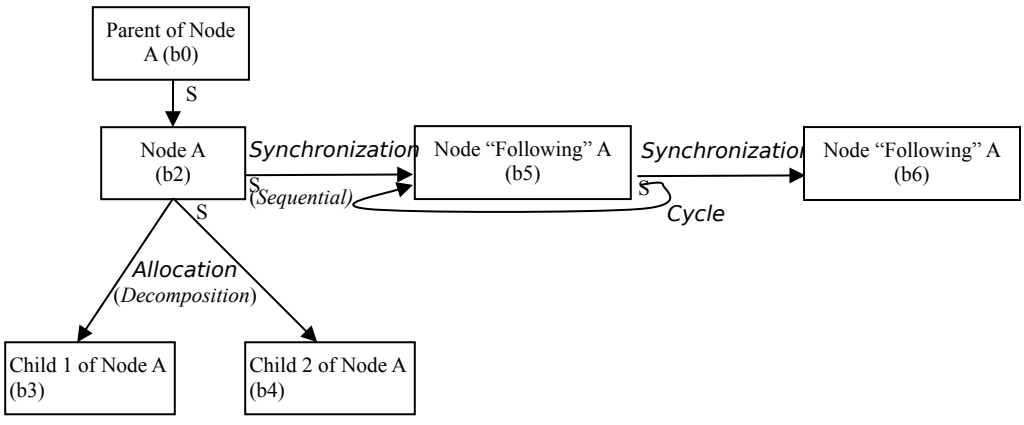


Figure 1. Behavior graph. (Relationships for Node A)

task-oriented behaviors. It can be thought of as a Directed Cyclic-Horizon Graph in which each node represents a behavior and each edge represents the link between two behaviors. A typical behavior hierarchy, centered on the node A, is shown in **Figure 1**.

In the behavior hierarchy, there are two types of directed edges generated from one node, namely *horizontal* and *vertical* edges. A horizontal edge represents a sequential relationship whereas a vertical edge represents a decomposition relationship. If *behavior A* is connected vertically to *behavior B*, we say that *behavior B* is a child *behavior of A*. Similarly, if *behavior A* is connected horizontally to *behavior B*, we say that *behavior B* follows *behavior A* or that *behavior B* is the follower of *behavior A*.

There are also two types of nodes in the behavior hierarchy: *internal nodes* and *leaf nodes*. Internal nodes have one or more outgoing vertical edges, thus they represent internal behaviors whose goals can only be achieved through the accomplishment of a set of sub-goals of their child behaviors. On the other hand, *leaf nodes* have no outgoing vertical edges and represent leaf behaviors that have an atomic goal and the execution required to achieve that goal.

Each node in the behavior hierarchy is an augmented connected graph quadruple $<B, H, V, b_0>$, where B is a set of task-achieving behaviors (vertices); H and V are sets of directed edges between behaviors ($H \cap V = \emptyset$); and $b_0 \in B$ is a behavior from which the execution begins. Each behavior $b_i \in B$ may have preconditions which enable its selection (the robot can select between multiple enabled behaviors), and termination conditions that determine when execution must be stopped.

In the behavior hierarchy a path along sequential edges, i.e., a valid sequence of behaviors is called an *execution chain*. H is a set of sequential edges that specifies temporal order of execution of behaviors. A sequential edge from b_2 to b_5 (**Figure 1**) specifies that b_2, b_3, b_4 must be executed before executing b_5. 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 root-to-leaf path through the behavior graph. Sequential edges may form circles, but vertical edges cannot; thus an ancestor behavior cannot be repeated.

During the behavior hierarchy traversal, currently executing robot's behaviors are stored in the *behavior stack*. The order in which items are pushed on and popped from the stack is defined by the BITE execution **Algorithm1** (*Section 3.2*). An *execution group* for a behavior is defined as a set of robots that are currently executing that behavior, or have the behavior on their behavior stack. Each robot belongs to the execution group of all behaviors that they have on their stack.

When the BITE execution follows a vertical edge in the behavior hierarchy, allocation is required. Similarly, when a horizontal edge is encountered synchronization is performed. In addition, synchronization is required for termination of robots execution of team behaviors. When a team behavior termination conditions are satisfied for a robot, BITE is triggered to coordinate the termination of this behavior with the other robots.

To allow BITE to **automate synchronization**, we impose a constraint on the semantics of multiple outgoing edges. Two outgoing sequential edges $< b_5, b_6 >$, $< b_5, b_2 >$ (**Figure 1**) signify a choice-point between alternative execution chain, i.e. a robot that finished execution of b_5 must choose either b_6 or b_2. When these execution chains are composed of team behaviors, the selection between alternatives must be coordinated, so that all (relevant) robots select the same execution chain. Therefore, BITE's synchronization services are triggered when multiple execution chains are enabled. Later on in the thesis complex teams and the identification of relevant team-members are presented.

To **automate allocation**, we impose a related semantic constraint on decomposition edges. Two outgoing decomposition edges $< b_2 , b_3 >$, $< b_2 , b_4 >$ (**Figure 1**) signify

complementary execution chains: execution chain beginning with b_3 and the execution chain beginning with b_4 must terminate for b_2 to be considered complete. By convention, vertical edges point only to the first behaviors of execution chains since in any case these behaviors must be executed before others in their respective chains. Similarly to the synchronization points, BITE's allocation services are triggered when multiple decomposition edges are enabled.

Given the above constraints BITE can easily determine synchronization and allocation points. 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. For all these points BITE must coordinate with the other robots through their own BITE processes.

Organization hierarchy

In order to be able to synchronize and allocate behaviors, BITE maintains information about robots responsible for the execution of team behaviors. This information is represented in the organization hierarchy (team hierarchy in [9, 10]), which is a Directed Acyclic Graph (DAG) whose vertices are associated with sub-teams of robots, and whose edges signify sub-team-membership relationships (**Figure 2**).

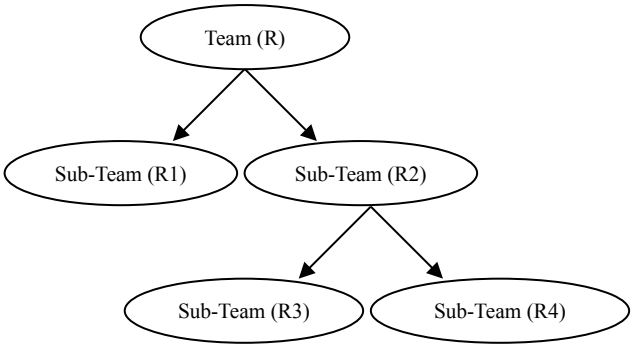


Figure 2. Organization hierarchy. ($R1 \subset R \cup R2 \subset R$, $R3 \subset R2 \cup R4 \subset R2$)

The organization hierarchy always has more than one node with complete set of robot team-members *R* at the root. The set is further broken down into robot sub-teams $R_i \in R$. To allow behaviors to determine which organizational unit is responsible for their execution, links are created between the behavior graph and the team hierarchy. A link from a *behavior Bj* to a *sub-team Ri* signifies that the *behavior Bj* is to be jointly executed by the *sub-team Ri*.

Social interaction behaviors

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 set of *social interaction behaviors,* which control inter-robot interactions and allow robots to dynamically create links between the behavior graph and the team hierarchy during execution of a particular behavior. Interaction behaviors typically control communication actions and execute interaction protocols (e.g., voting) that govern coordinated activity and are responsible for status notifications between robots. For instance, a simple interaction behavior for the synchronization can be decomposed into four atomic interaction behaviors that are executed in a sequence: announce vote, send votes, tally votes, and announce the selected winner.

There are three different types of social interaction behaviors: a) synchronized selection of behaviors prior to their execution; b) allocation of robots to behaviors; and c) synchronized termination of behavior execution.

Synchronized selection is a protocol that allows a team of robots to synchronize the selection of the next horizontal behavior to be executed in the behavior hierarchy graph. *Allocation* of robots to behaviors occurs when a behavior in the graph is to be decomposed into children behaviors. If a node in the graph is decomposed into *n* behaviors, the allocation protocol is executed to divide the robot team into *n* sub-teams of robots to execute each behavior.

During the execution of vertical edges in the behavior hierarchy, each behavior is executed in parallel by a team of robots and there is no need for synchronization. On the other hand, for horizontal edges each robot must terminate the particular behavior in order to proceed with the next synchronized selection. Therefore, before calling the synchronized selection, the s*ynchronized termination* of the behavior execution is performed to determine which robot finished execution of the behavior. For instance, a parent behavior has several robots doing a distributed search for a target. A robot that finds the target terminates the search and informs its teammates. When all robots terminate their search the parent behavior terminates.

Behavior hierarchy, organization hierarchy and social interaction behaviors are all interconnected. Interaction behaviors are represented with the same behavior graphs as the task-oriented behavior of the robots. Thus synchronization, allocation, and termination points in the behavior graphs of interaction behaviors are linked to other interaction behaviors, creating hierarchical interactions.

Figure 3 shows an example of the three fundamental structures of the BITE system described in this chapter: **a)** organization hierarchy, **b)** behavior hierarchy, and **c)** social interaction behaviors. We will use it to demonstrate a simple voting interaction behavior composed of four atomic interaction behaviors: announce vote, send votes, tally votes, and announce selected winner.

The organization hierarchy in the **Figure 3a** is composed of three robots identified as *a*, *b* and *c*. Each robot team is linked with its associated behaviors. Similarly, the behaviors are linked with their associated teams, which can be either predefined or linked dynamically during the BITE execution.

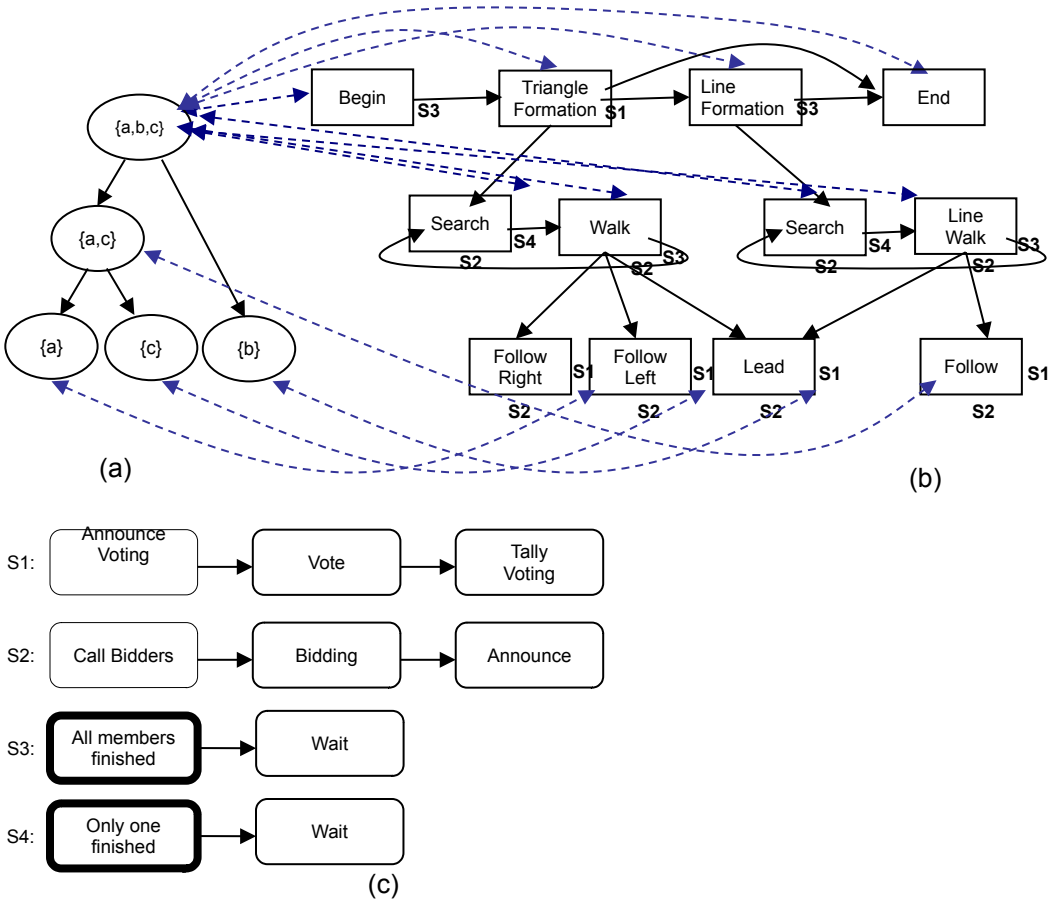


Figure 3 BITE Structures and Links for a formation maintenance task**.** (a) Portions of the team hierarchy, (b) behavior graph, and (c) social behaviors. Links from (b) to social behaviors in (c) are denoted by S1–S3.

Figure 3b shows an example of a simple behavior graph, constructed for multi-robot formation maintenance tasks. There are two formation behaviors here - *Triangle Formation* and *Line Formation*. Execution begins with triangle formation, and can (under specific conditions) switch to the line formation. Both formations use the behavior *Search* to choose a role in the formation (leader, follower, etc.). Once their role is selected, robots choose between the *Walk* behavior (walking in triangle) and the *Line Walk* behavior (robots

follow each other in a line). The *Walk* and the *Line Walk* behaviors are marked as team behaviors, and require two important teamwork capabilities:

- Synchronization - to guarantee that all robots select the same behavior, and start/end the behavior at the same time
- Allocation - to guarantee that only a single leader for the formation behavior is chosen

Figure 3c shows four social behaviors: a voting behavior *S1*, bidding behavior *S2*, termination behaviors *S3* and *S4*. Behavior *S1* is a *synchronized selection* behavior, where one predetermined robot announces the call for votes and the candidate behaviors. Then it collects the votes from all team members and announces the winning behavior. This behavior is then selected for execution by each robot in the team.

Behavior *S2* is the *allocation of sub-teams to behaviors* and represents the sequential phases of a market-based protocol to be used in allocating the children behaviors to different robot sub-teams. Once the allocation is made, appropriate links are created between the allocated behaviors in behavior-graph and the sub-teams in team-hierarchy responsible for executing them.

Behaviors *S3* and *S4* are very simple *synchronized termination behaviors,* which are used in the formation task. These behaviors are marked bold because at this point all robots constantly communicate with each other to update their status. For example, a robot that has terminated execution of a joint behavior waits for all the other robots to reach the end of their execution chains as well, before they all begin their joint execution of a new behavior.

Let's assume that three robots are executing the task *Triangle Formation (***Figure 3b***)*. They have together finished execution of the behavior *Search* and have started on *Walk*. The robots must jointly decide how to allocate different roles of the formation among

themselves. 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 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. Two sequential transitions connect the behavior *Search:* one to the *Walk* behavior, and the other to the *Line Walk* behavior. A synchronized decision is to be made on which connection to choose such that all robots select the same behavior and the execution must begin simultaneously. The social behavior S1 is used to coordinate this synchronized selection

3.2. Algorithms

Algorithms presented in this section are the base algorithms of the BITE system. The main *Principal Control* algorithm is responsible for the behavior graph and for the execution of all behaviors. The *Resource Precondition Control* algorithm is responsible for recourses and it is linked to the platform in which BITE is executed. The *Fuse Information* algorithm is responsible for linkage between all team members and it is based on the team hierarchy. Finally, the *Operation Control* algorithm allows an operator to take control over a robot for its own use.

3.2.1 Algorithm 1 – Principal Control Algorithm

This is a basic algorithm that controls the coordinated execution and selection of behaviors by robots. The BITE control loop traverses the behavior hierarchy graph from root to leafs, simultaneously executing each node and its selected children. The behavior on top of the *behavior stack* corresponds to the currently active behavior of a robot in the robot-team. Given a set of robots that is currently executing some node in the hierarchy (i.e. synchronized), several important events can occur which trigger transitions in execution.

If a given set of robots is executing a leaf node, they will simply continue execution of that node until the termination condition of one of the behaviors in their stack is met. At this point, all robots either move to the parent of the terminated behavior, or move horizontally to the following behavior if it exists. In both cases, once a robot transitions, it enters a waiting state in which it will not begin execution of the next behavior until all robots in the execution group of the terminated behavior have transitioned as well. This ensures that all robots will begin executing the next behavior simultaneously, which is essential if the next behavior is an internal behavior.

Algorithm 1 CONTROL

Input: behavior graph $< B, H, V, b_0 >$, 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. , $< s_0 , b_0 >$ 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) check (and wait for) resources for b_i (Algorithm 2)
   (b) execute b_i until it terminates
   (c) while $b_i \neq$ *top(G),pop(G)*.
   (d) break parallel execution, go to 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.,* $< b_0, s_i >$ is a sequential edge
   (b) if Q is empty, go to 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, go to 3.

On the other hand, if a set of robots is executing an internal node, all robots at that node execute their social behavior until each one of them has enough information to determine the child behavior that it should transition to.

Each robot executes **Algorithm 1** using its own copy of the three BITE structures (behavior hierarchy, organization hierarchy and social behaviors). Executions begin when the initial behavior of the behavior graph (typically the root) is pushed on the behavior stack (*lines 1-2*). Then the algorithm loops over the four following phases:

(i) Recursively expand the children of the behavior, allocating them to sub-teams if necessary *(lines 3a-3c)*.

(ii) Execute the behavior stack in parallel, waiting for the first behavior to announce its termination *(lines 4a-4d)*. All descendants of the terminating behavior are popped off the stack (i.e., their execution is also terminated *line 4b*).

(iii) Next a synchronized termination takes place *(line 6)*, which can in result in one of the following:

(iv) Allocation of a new behavior within the current parent context, in which case this behavior will be pushed on the behavior stack (line 7).

(v) The robot chooses between any enabled sequential transitions from the terminated behavior (lines 8a-.8e). This process normally results in new behaviors being pushed on the stack and the 'goto' line 3 causes the recursive expansion and allocation to sub-teams (line 9).

The recursive allocation of children behaviors to sub-teams on *lines 3a-3c* relies on the call to the *Allocate()* procedure. To make the allocation decision, *Allocate()* takes the current execution context (i.e., current stack, available children), and then calls the

appropriate social interaction behavior for descendants of the current node. The current execution stack is used to ease allocations, 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 only composed of a single robot).

Synchronized termination (*lines 5-7*) and selection (*lines 8a-8e*) rely on calls to the procedures *Terminate() and Decide()*, respectively. *Terminate()* is responsible for invoking the termination interaction behavior, which can return a new child behavior for execution under the current node.

If the current node does not have descendants, the next behavior in the execution chain must be selected by the *Decide()* procedure, which calls the appropriate 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 in which no more behaviors are available in the execution chain. This case signals a termination of the execution chain, which in turn signals termination of the parent, thus looping back to *line 5*. For clarity, we omitted the check whether a parent actually exists, in which case the end of the behavior graph has been reached and the execution halts.

3.2.2 Algorithm 2 – Resource Precondition Control Algorithm

In BITE, as in other behavior-based architectures, the execution of all selected behavior is done in parallel (**Algorithm 1**, step 4). However, in contrast to other behavior-based architectures, our architecture has a notion of resource preconditions that prevents currently executing behaviors to utilize an already utilized resource (**Algorithm 2**). If a resource is not available, the behavior takes a standby position until the resource is released (***Example Section 3.4***). Semaphores are used to prevent deadlocks.

Algorithm 2 RESOURCEPRECONDITIONCONTROL

Input: interaction behaviors set *O* , resource status *R*

1. Resource precondition:
   (a) *p* ←*getResourcePrecondition(O)* *// return resource precondition*
   (b) if *ResourceStatus(R, p)*= true, goto 2. *// Boolean function*
   (c) else: *SleepingWait ()* and goto 1. *// after 128ns return from sleep*
2. Execute behavior *O*.

3.2.3. Algorithm 3 – Fuse Information

The Fuse Information algorithm is designed and intended to run concurrently with the *Resource Precondition Control* algorithm (**Algorithm 2**). The algorithm is designed to coordinate the linkage between all team members and it is based on the team hierarchy.

In the first phase of the algorithm (**Algorithm 3**, *lines 1a-1c*), each robot determines whether new information has become available. This information, such as newly satisfied conditions, affects the robot's current behavior stack. That potentially affects 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. This is done through the *Inform*() procedure, which refers to an appropriate social interaction behavior. It takes into account the cost of sending a message versus its utility as is done in TEAMCORE [9].

Algorithm 3 FUSEINFORMATION

Input: behavior graph < B, S, V, b_0>, 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 (< b, f > a sequence edge) is satisfied, *Inform(b ,t, O)*
2. for all teams t in the team hierarchy:
   (a) $C \leftarrow \{b \mid b \in B$, 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 (< b, f > a sequence edge) is satisfied, *Inform(b ,t, O)*

As stated earlier, a robot is strictly responsible of informing only its own team-members. The second phase of the algorithm (**Algorithm 3***, lines 2a-2b*) allows a robot to determine whether the newly acquired information is also relevant to other sub-teams that the robot is a member of. The *Inform()* procedure decides whether the robot should provide the information to these sub-teams or not.

For instance, suppose a team of robots is executing the *Triangle Formation* (s**ection 3.3**) and are currently executing the *Walk* behavior. Suppose now that one of the follower robots is currently executing the *Follow Left* behavior. **Algorithm 3** guarantees that when the robot meets termination conditions of *Follow Left*, *Walk*, or *Triangle Formation*, it informs its team members. In addition, if the robot receives a termination condition from its leader, it informs members of the sub-team associated with leader.

3.2.4. Algorithm 4 – Operation Control Algorithm

The *Operation Control* algorithm (**Algorithm 4**) follows up all the events that take place when an operator intervenes in the work of a robot. Below we present a line by line description of the algorithm.

- When the operator wants to control a robot, a message is sent.
- When a robot receives the message (*line 1*) it must signal to itself and to all its teammates (*line 2b*) that it is controlled by an operator (*line 2a*).
- When the robot is released by the operator, it notifies its teammates again and its status changes (*line 3a-3b*).
- The change of status of the robot is executed in parallel for all its behaviors (*line 4*).
- Every launched behavior is set to OprWait status until the operator releases control of the robot.
- When a robot receives a message that a *Robot A* is controlled by an operator, it saves this message in its stack until it receives a message that the *Robot A* is released (*line 5*).
- While executing social behaviors (*line 6*), BITE runs as usual. When a robot is released, it doesn't resume its execution, but operates at the exact point in the behavior graph where currently its team is executing.

An important point is that members under the operator control are automatically excluded from the usual decision-making process, but they do keep track of the execution and decisions of their team members. Therefore, when they are released, they 'know' what their team members are currently executing and what their future tasks will be.

Algorithm 4 OPERATIONCONTROL

Input: behavior graph < B, S, V, b_0>, team hierarchy T, interaction behaviors set O, behavior stack G

1. $t \leftarrow$*TeleoperatorStatus*() *// notification*
2. if t= *true* and *OPR_END=GetOperStatus()* then
   a. *SetOperStatus(OPR_BEGIN) //change indicator*
   b. *SendMsgToAllTeam(OPR_BEGIN)//send to all members of team about my transition*
3. if t= *false* and *OPR_BEGIN=GetOperStatus()* then
   a. *SetOperStatus(OPR_END)) //change indicator*
   b. *SendMsgToAllTeam(OPR_END) //send to all members of team my transition*
4. Executed in parallel for all behaviors b_i on G:
   a. if b_i== *Executed* and t= *true* then
      i. *ToReleaseResource()*
      ii. *ChangeStatus(OprWait)*
   b. if b_i== *OprWait* and t= *false* then
      i. *ChangeStatus(Executed)*
5. if received message "operation control" from *robot A*
   a. push *robot A* onto OprWaitMembers stack W.
6. **Social behavior :** *a* doesn't participate in a choice.

are expected to be selected in the future, thus allowing robots to anticipate the needs of their teammates [8]. We have found it useful to run a proactive communications algorithm that informs teammates of sensed knowledge that may be relevant to them. The relevance is chosen based on the team hierarchy and the behavior graph.

3.3 A Simple Execution Example

This chapter presents an example of a BITE execution. It demonstrates execution of the Triangle Formation by three *Robots A*, *B*, and *C* (**Figure 5**). In the Triangle Formation robots must walk in a triangle, where one robot is leading and the other two are following – one on the right and the other on the left. All robots have a different color painted markers on their back: *Robot A* is pink, *Robot B* is blue and *Robot C* is white. In the formation robots maintain their positions and head angles relative to the leading robot. Color segmentation is used to identify the angle and AIBO's distance sensor (infra-red) is used to maintain distance within some constraints.

All three structures of the BITE architecture that are used for this example are shown in **Figure 4**. The top left graph represents the organization (team) hierarchy, the top right graph shows the behavior hierarchy and at the bottom social interaction behaviors are shown. Each robot has a copy of this architecture.

Initially, the behavior stack of each robot contains only the *StandUp* behavior (**Algorithm1***, line 1-2*). Therefore, the state of the team at the startup is as follows (**Figure 5a**):

Robot A = StandUp

Robot B = StandUp

Robot C = StandUp

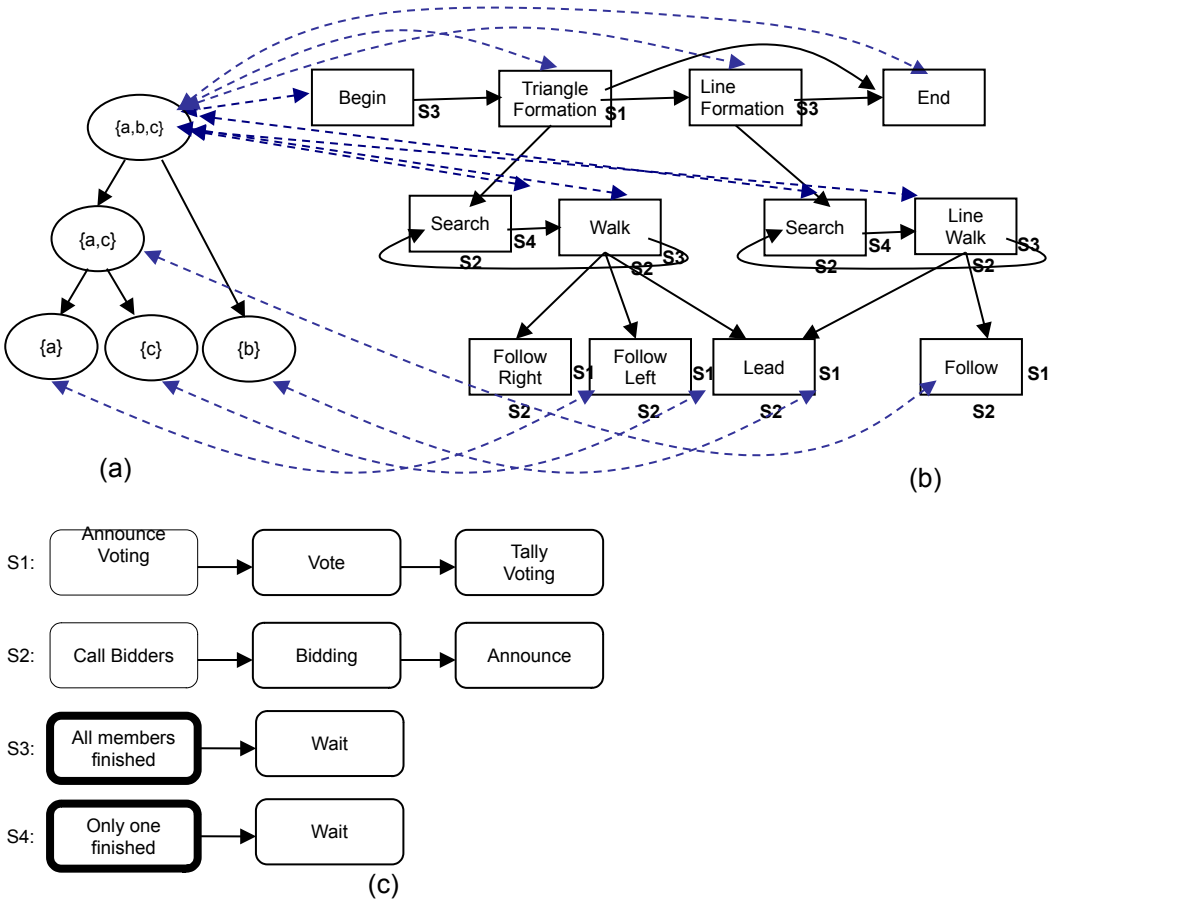


Figure 4 Triangle Formations.

Three structures of the BITE architecture used in the *Triangle Formation* example. The top left graph represents the organization (team) hierarchy, the top right graph shows the behavior hierarchy and at the bottom social interaction behaviors are shown.

StandUp is an atomic node in the behavior graph (**Algorithm 1,** *line 3*), therefore its execution must be synchronized. Upon the termination of the *StandUp* behavior, each robot

pops out the *StandUp* behavior from its behavior stack (**Algorithm 1,** *line 5*) and executes the social behavior *S1* (Wait All) (**Figure 5b**).

(a) **(b)** **(c)**

Figure 5. Executed *StandUp* behavior.

Start execute behavior *StandUp* (a), run behavior *StandUp* (b) and finish execute behavior *StandUp* (c)**.** *Robot 1(A)*-pink, *Robot 2 (B)*-blue and *Robot 3(C)*-white.

For example, if *Robots A* and *B* are first to terminate execution of the *StandUp* behavior, the state of the team is:

Robot A = empty

Robot B = empty

Robot C = StandUp

StandUp behavior is executed simultaneously by all teammates, as shown in the behavior hierarchy graph in **Figure 4**. This behavior is linked to the team hierarchy, and points at all team members (*A*, *B* and *C*). Therefore, when terminating, each robot must send a message of termination to the entire team, and wait for termination messages from the others.

When all three robots terminate the *StandUp* behavior and notify others, they have only one option to choose from the behavior graph. Thus the next behavior that is pushed on the behavior stack is *Search* (**Algorithm 1,** *line 8c*) and the state of the team is:

Robot A = Search
Robot B = Search
Robot C = Search

Unlike the *StandUp* behavior, the *Search* behavior node has children (subtasks) in the behavior graph; therefore the social behavior *S2* (Team-wide allocation of robots and sub-teams to behaviors) is launched (**Algorithm 1,** *line 3*). *S2* will decide the next actions of each robot, taking in to account the preconditions and the bidding of each robot. Based on the links from the organization (team) hierarchy, each robot will choose its behavior and start the execution (**Figure 6**). For each robot, two parallel behaviors (*Search* and *Search 'color'*) are launched (**Algorithm 1,** *line 4*) and the state of the team is (**Figure 7a**):

Robot A = Search → Search blue (the blue color paper)
Robot B = Search → Search pink
Robot C = Search → Search pink

Before executing the *Search 'color'* behavior robot launches *Resource Precondition Control Algorithm* (**Algorithm 2**) to see which resources are required for this behavior and whether these resources are available. When two or more currently executing behaviors use the same robot's resources (for instance leg resources for walking), a conflict is created. **Algorithm 2** allows these behaviors to dynamically schedule their control of the resources and run in parallel, each taking control of the resources only when needed.

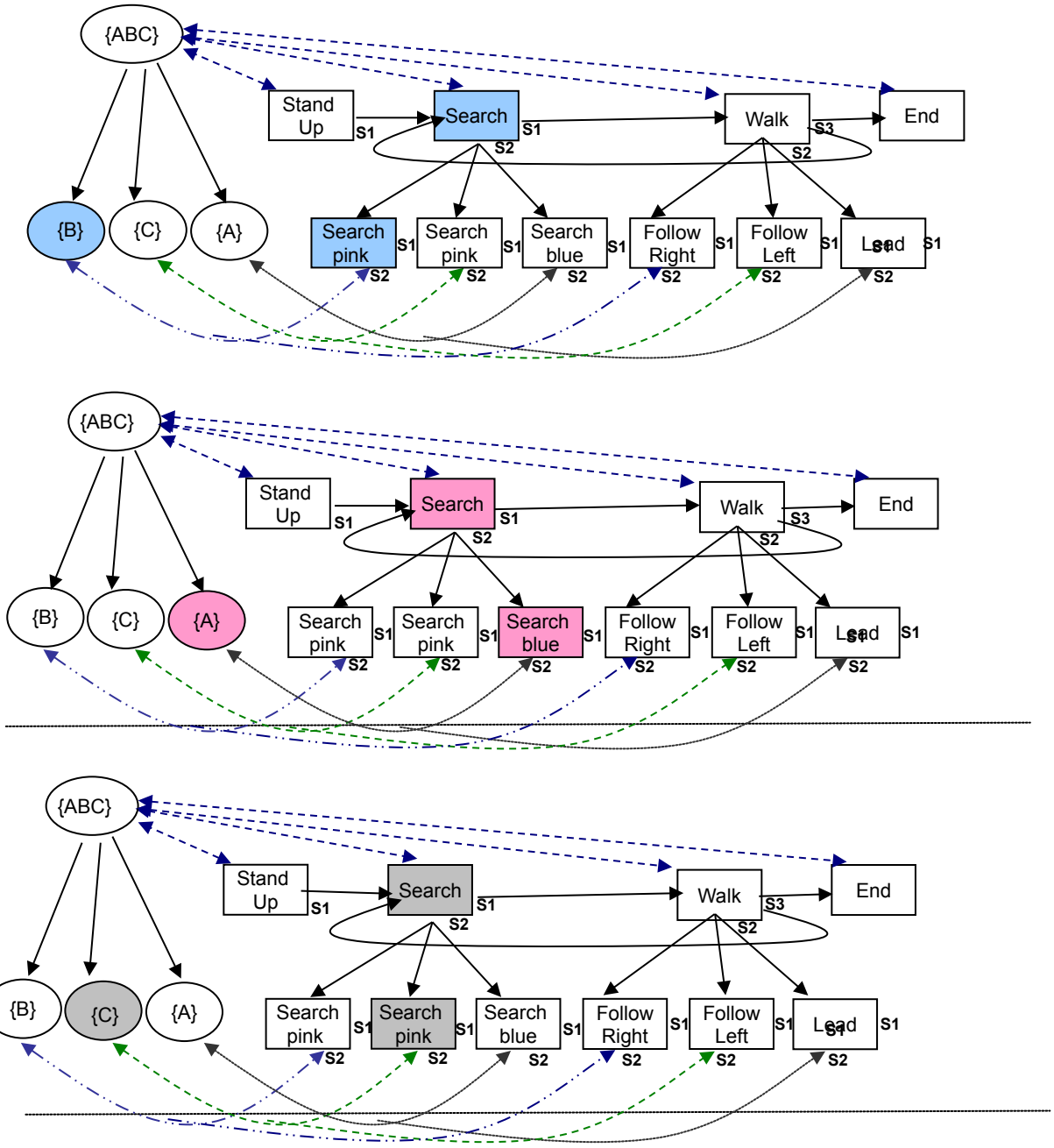


Figure 6. State of the team.

(Top) *Robot B* execution social behavior *S2*
(Middle) *Robot A* execution social behavior *S2*
(Bottom) *Robot C* execution social behavior *S2*

Each robot now executes its own task, and there is no communication between the team-members until they finish the *Search 'color'* behavior (**Figure 6**). For example, when *Robot B* finds its designated color (pink), it finishes the behavior. It does not launch any new social behavior but returns to the *Search* behavior and sends a message to all the teammates stating that it successfully terminated the *Search 'color'* behavior (**Algorithm1**, *line 5-7*). If *Robot B* and *C* finished their *Search 'color'* behavior but *Robot A* is still searching, the state of the robots is:

Robot A = Search → Search blue (the blue color paper)

Robot B = Search

Robot C = Search

Since *Robot A* is still searching for the blue color, *Robot B* and *C* start the S1 social behavior and wait for their teammate to finish its search. When *Robot A* successfully terminates its *Search blue* behavior, it returns to the *Search* behavior and notifies its team members (and itself).

If the number of messages received by each robot is the same as the number of robots in the team, the team transitions to the execution of the next behavior *Walk* (**Figure 4**) and the state of the robots is:

Robot A = Walk

Robot B = Walk

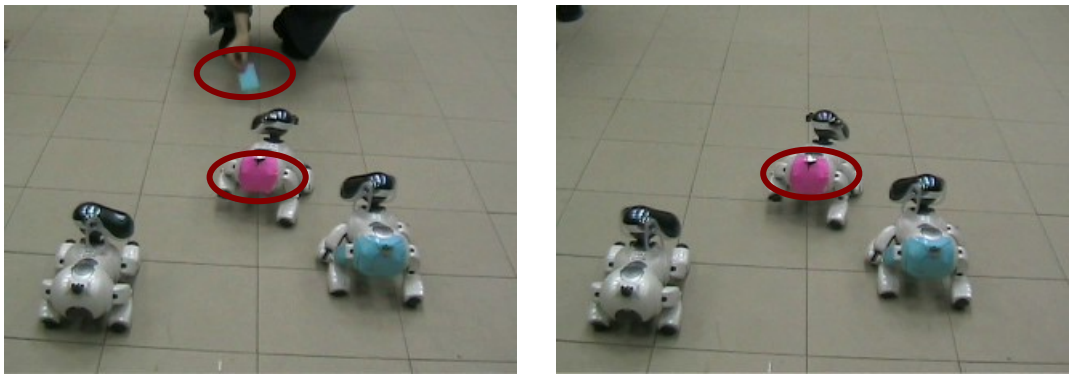
Robot C = Walk

Execution of the *Walk* behavior is similar to the execution of the *Search* behavior in the sense that robots go through the same allocation and synchronization process of bidding, executing and waiting. The difference is that the robots have to execute the social behavior *S2* (Call Bidders) and bid on their position in the formation. The bidding depends on the color they were searching in the *Search 'color'* behavior and on the angle of their head when the search was completed. Therefore, in our example the state of the robots is:

Robot A = Walk → Lead

Robot B = Walk → Follow Right

Robot C = Walk → Follow Left



(a) **(b)**

Figure 7. Position of each robot in Triangle Formations.

(a) All robots execute the *Walk* behavior.
(b) All robots terminate the *Walk* behavior.

During the *Walk* behavior execution, *Robot A* leads and follows the blue paper, *Robot B* follows *Robot A* on the right and *Robot C* follows *Robot A* on the left (**Figure 7a**). The robot's sensors monitor the distance between them and the color that they must follow (the color they searched in the *Search* behavior). Robots continue executing the *Walk* behavior only because all of them sense their color. When a robot in the formation stops to sense its designated color, it stops and terminates its *Walk* behavior and notifies itself and its members. Robots in the formation that receive the termination message from their teammate also stop walking and terminate their *Walk* behavior (**Figure 7b**) and the state of the team is:

Robot A = empty

Robot B = empty

Robot C = empty

At this point robots execute the social behavior *S3* (Leader pick). The leader of the *Walk* behavior (*Robot A*) randomly chooses the next execution step, which can be either *Search* or *End* behavior (**Figure 8**). When the choice is made, the leader notifies its team members. If the choice is to execute the *Search* behavior, then the process described above repeats; otherwise the BITE execution terminates.

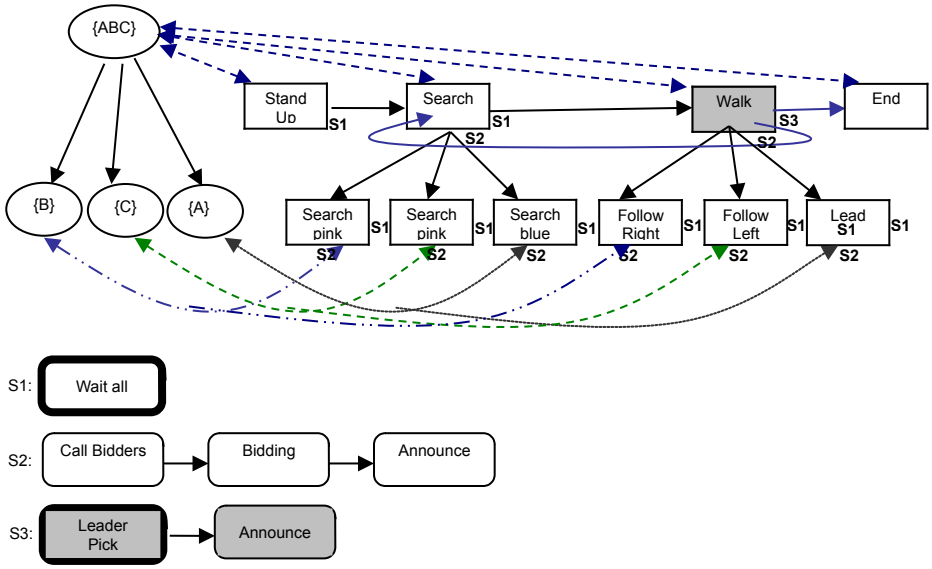


Figure 8. Termination of *Walk* behavior execution by all the robots.

Chapter 4

Hardware Configuration and Software Applications.

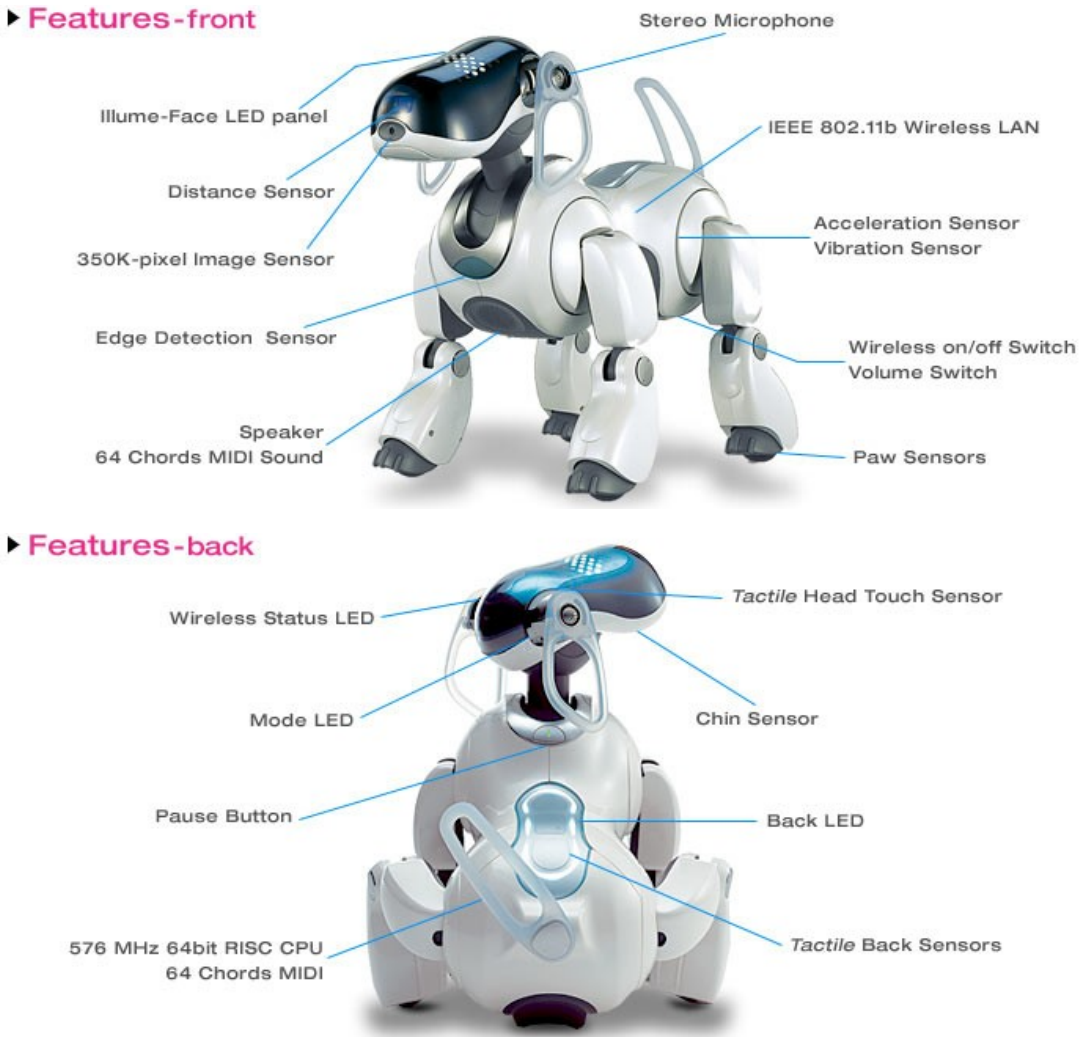


Figure 9. Aibo front and back features.

4.1 The AIBO

AIBO is the name of the robot developed by Sony Company for the entertainment market. The robot mimics a dog and it has four legs, a tail, it can bark and make many other sounds. From a technical point of view the AIBO is an impressive engineering achievement. The AIBO has a MIPS Little Endian CPU sufficiently powerful to perform tasks such as on-line image and voice recognition, a video camera, distance sensor, several buttons, pressure sensors and step based engines that are not only capable of offering resistance to change but also of reading their current position (**Figure 9**).

From the system's point of view the robot is composed of several objects: sensors, parts and the body. We didn't use the body object for our experiment. The main sensors that were used for our system are the *Distance Sensor* and the *Image Sensor* (**Figure 9**). *Distance Sensors* can be attached to parts of the robot or to the robot's body. *Image Sensors* are programmed to recognized objects by their color (pink or blue).

A robot is composed of a set of movable parts. Each part can be moved according to three parameters*: tilt, pan* and *roll*; and has a set of operational limits that indicate the extent to which the part can be moved to.

The robots used for our experiments also had an abstract *Motion Controller* that abstracts the system from the task of controlling the robots motion. It provides a way for the system to indicate the speed at which the robot should move and takes care of all the necessary tasks. For instance, a *Motion Controller* for the AIBO is capable of controlling its legs to make it walk in the desired direction.

4.2 Tools

Sony provides several tools that can be used in the development of applications for the AIBO. These alternatives range from low level applications that require a full control over all variables of the system, to high level ones that only allow scripting of behaviors. We added other alternatives to the tools provided by Sony, such as OPEN-R and Tekkotsu.

4.2.1 OPEN-R

From a Software Developer's point of view OPEN-R is a C++ Application Programming Interface (API) that defines how the objects of the system should work. OPEN-R facilitates a new way of developing software because OPEN-R also defines how the objects interact with each other. Unlike a normal operating system which developers are used to, OPEN-R does not provide abstractions such as processes and threads of execution. It provides an abstraction of a message passing mechanism in which producers and consumers only run when the recipients of the message are ready to receive the message or when a message has been sent. This makes developing software in OPEN-R harder but it also has its advantages.

The approach taken in OPEN-R seemingly has one main goal: to prevent CPU use if there isn't any input from the sensors. That is, from the OPEN-R perspective it only makes sense to use CPU when the systems sensors produce data and the same principle should be applied to higher levels of abstraction. For instance, if the system has a face recognition module, the module computes only when the digital camera produces an image or a set of images. After the face recognition module has detected a face, the logic that needs to know when a face is recognized can start running.

By developing AIBO Sony attempts to make its OPEN-R API a standard for the development of entertainment robot systems. Sony states that [15] "OPEN-R provides a

layered approach that is optimized to enable efficient development of hardware and software for robots." OPEN-R is well suited for low level control of the robot. High level tasks, like walking, are harder to control since it involves monitoring several engines and reading sensors of all robots legs. Unfortunately, OPEN-R does not provide an implementation for such a high level task. In addition, OPEN-R's message passing mechanism is hard to control and involves changing several files. However, there are several publicly available modules for OPEN-R that provide a high level control over certain robot functionalities. Unfortunately, combining modules developed by different software engineers in the same robot is challenging.

4.2.2 Tekkotsu

Tekkotsu was created at the Carnegie Mellon's University labs. Tekkotsu means "iron bones" in Japanese and is used often in the context of a buildings' structural framework [11]. In a similar manner, the Tekkotsu framework provides a skeleton that abstracts the AIBO's internals and that can be used to create complex behaviors and applications.

On its basis the Tekkotsu framework is a C++ layer on top of OPEN-R that provides a greater abstraction of low level details of the robot. Tekkotsu pursues the ambitious goal of being an application framework for the development of robotic platforms. Currently it supports OPEN-R enabled robots which include AIBO's ESR-210, ESR-220 and ESR-7 models.

Tekkotsu is an object oriented event passing architecture that is similar in the design to Java. In Java when an object provides notifications, it dispatches events to its listeners. This means that objects can register themselves as listeners of events generated by another object. Similarly, in the Tekkotsu framework it is possible to register a listener that is notified when a specific button is pressed, or when the digital camera has taken another

picture, or when the distance sensor installed on the robot's head has taken another measure. The main goal of this approach is: 1) simulate what the OPEN-R API provides but with a higher level of abstraction; 2) avoid the spaghetti code that tends to be written when using OPEN-R, because OPEN-R does not support function calls.

Tekkotsu lies above OPEN-R in terms of abstraction. It provides a high level functionality needed for the robot to start walking and to move its head, but it also possesses the possibility to access the low-level details of the robot. Another possibility offered by the framework is to register a listener that is notified when sensor information is ready, which makes even easier to obtain low level information.

4.3 Project's Choice

For this project the Tekkotsu framework was chosen for the development of the interface with the AIBO for several reasons. First, the Tekkotsu framework provides access to the low level detail of the robot using a high level API. Its abstraction allows convert the notification of new sensor information into a native C++ method call. This in turn enables the acquisition of values of the sensors available in the AIBO without using complicated message passing mechanism of OPEN-R.

Besides providing an abstraction for the low-level details of the robot, Tekkotsu also provides high level controls for complex operations. Tekkotsu modules are organized as behaviors and sub-behaviors. For example, walking is a sub-behavior that allows robot to walk simply by indicating the desired target speed. The speed can be either forward speed, roll speed or side speed.

OPEN-R provides sockets to enable communication with an external computer system. Tekkotsu builds upon this service and creates abstraction that makes it easier to create a service to provide and obtain data from external systems, be it another AIBO or an

external computer system. This is crucial to provide an external monitoring and control interface that can act both as another user input device as well as to present, in a human readable fashion, the structures maintained by the system.

Tekkotsu depends solely on OPEN-R. This is a great advantage since OPEN-R enables objects to be moved from the AIBO to an auxiliary system. By depending only on OPEN-R, Tekkotsu also shares the advantage that enables ultimately to move heavy operations to an auxiliary system that runs concurrently with the AIBO.

4.4 BITE

The BITE architecture is based on Tekkotsu and OPEN-R which allows to use Tekkotsu's abstract skeleton for the AIBO's internals and to create complex behaviors (**Figure 10**). In addition, the architecture is platform independent and allows intervenient execution of multi-platform robots (ESR-210, ESR-220 and ESR-7 models).

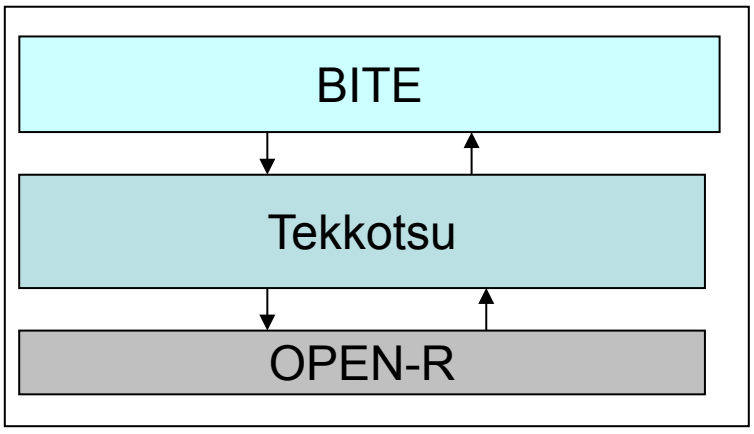


Figure 10. Layer diagram.

The BITE framework is a C++ layer on top of the Tekkotsu framework and provides the ability to use one robot as well as multi-robot teams. Control of multi-robot

teams is done through a communications system described in the application layer. The application layer is build on top of the Transmission Control Protocol layer, which is proved by the OPEN-R and Tekkotsu.

4.4.1 BITE Installation

The BITE system can be compiled and run only on UNIX or Linux computers. First, copy the BITE project to your directory then configure and run the Makefile. To configure the Makefile you need to set the **MEMSTICK ROOT** to the path were the robot's memory stick is mounted on the UNIX or Linux system.

You have to generate binaries for Tekkotsu that run on either ERS-210, ERS-220 or a specific model. This is done by creating a file named ***TARGET MODEL*** in the Tekkotsu installation directory and indicating one of the following:

TGT ERS210 — for the ERS-210 model.

TGT ERS220 — for the ERS-220 model.

TGT ERS2xx — for both the ERS-210 and ERS-220 models.

TGT ERS7 — for the ERS-7 model.

You also need to setup the wireless interface for the Tekkotsu Monitor tool. This can be achieved by creating the ***project/ms/open-r/system/-conf/wlandflt.txt*** file (see the Open-R documentation). You have to add the IP address of all robot team members to enable communication between them. This can be achieved by creating the ***project/ms/open-r/system/-conf/team.txt*** file with this content:

name=hostname\t IP=132.70.5.83

After compiling and configuring all required files, insert the robot's memory stick into the reader on the UNIX or Linux machine. In the command prompt type either '*make YourMountDriveName*' or '*make update'* commands. Then insert the stick into the AIBO

robot to start it up. This will cause the AIBO to boot in an emergency stop (see Tekkotsu documentation). Next start the monitor tool of the Tekkotsu located in the ***/usr/loca/Tekkotsu/tools/mon*** directory. Finally, to open the controller graphical user interface type: ***./ControllerGUI*** "***Aibo IP address***" (requires Java 1.4 or higher installed on your system). To make the AIBO respond to your commands, double tap its back button, which will cause it to exit the emergency stop procedure.

Chapter 5

BITE: Experiments, Results and Conclusions.

5.1 Experiments Setup

The experiments of the BITE system were conducted on the AIBO ESR-7 platform described in chapter 4. We run 50 experiment sessions with the main purpose to test the influence of various social behaviors upon performance time and failure handling ability. All experiment sessions were carried out in the same environment and identical conditions, using three robots.

The behavior graph designed for our experiments had twenty-one behaviors and nine social behaviors. It had two allocation points and three synchronization points. We ran three separate versions of the tasks, where the same behavior graph was used with different

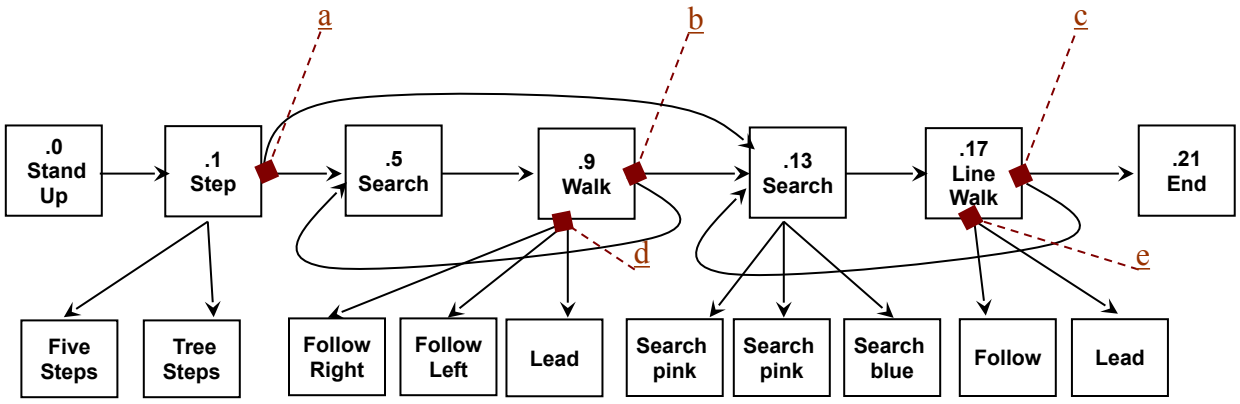


Figure 11. Behavior graph.

Nodes represented behaviors.

Points A, *B*, *C*, *D* and *E* represented social behaviors. Each point can be one of the social behaviors shown in the **Table 1.**

combinations of interaction behaviors. During all experiment sessions, the designed behavior graph remained static. The various social behaviors (**Figure 11**) differed in their use of connection between the teams, as well as in the *allocation social behavior* and in the *synchronization social behavior*.

The basic mission of each session was to:

1. Position in a triangle.
2. Walk a certain distance in the triangle formation.
3. Line up and cross the room from one wall to another opposite wall.

Table 1 displays all social behavior profiles used for the experiments, which are triggered at each of the *A, B, C, D, E* points in the behavior graph in **Figure 10**. The table shows the type of a social behavior, usages of the connection, and description of the basic activities. Note that only certain social behaviors can be executed from a given behavior.

SB Name	Transition Type	Use Network	Can be executed from behaviors	Description
SF1	Synchronization	No	1, 9, 17	Choice is made based on robots resources and ID
SF2	Allocate	No	9, 17	Choice is made based on robots ID
SF3	Allocate	Yes	9	Choice is made based on the robots head angle
SF4	Synchronization	Yes	17	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	1, 17	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	17	Choice is made based on the robots color
SF7	Synchronization	No	9	Robots decide on the next behavior based on the number of times this behavior is executed
SF8	Synchronization	Yes	1	Each robot sends its bid
SF9	Synchronization	Yes	9	Each robot sends its bid number. Then (the average of all sent numbers by all team members) MOD (number of branches from the current behavior) is computed.

Table 1. Social behavior profiles used for the experiments.

5.2 Experiments Data

All experiments were divided into nine groups based on which behavior nodes in the graph were traversed and which social behaviors were executed by the robots in the experiment (**Table 2**). As stated earlier, the transition points *A,B,C,D,E* in the behavior graph mark transition points where one of the nine social behaviors was used; where *A, B* and *C* are *Synchronization* points, and D and E are *Allocatio*n points. Each group experiment was executed five to seven times. We measured synchronization and allocation times, as well as overall task time.

Group name	Executed behaviors	Executed social behavior point A-B-C-D-E	Number of experiment
Group 1	0-1-5-9-13-17-21	SF1-SF1-SF2-SF1-SF2	6
Group 3	0-1-5-9-13-17-21	SF1-SF9-SF2-SF1-SF2	7
Group 4	0-1-5-9-13-17-21	SF5-SF1-SF2-SF1-SF2	5
Group 5	0-1-5-9-13-17-21	SF8-SF9-SF3-SF1-SF2	7
Group 7	0-1-5-9-13-17-21	SF5-SF1-SF2-SF5-SF2	5
Group 8	0-1-13-17-21 (executed 13-17 four times)	SF1-SF9-SF2	5
Group 9	0-1-13-17-21	SF9-SF9-SF2	7
Group 2	0-1-5-9-13-17-21 (executed 5-9 twice and 13-17 four times)	SF1-SF9-SF2-SF1-SF2	5
Group 6	0-1-5-9-13-17-21 (executed 5-9 four times)	SF1-SF7-SF3-SF4-SF6	5

Table 2. Experiment group type is based on which behavior nodes in the graph were traversed and which social behaviors were executed by robots in the experiment.

As can be seen from the behavior graph in **Figure 11**, transition 5-9 and 13-17 have a cycle, which allows robots to loop back in the execution chain. Thus groups 2, 6 and 8

have repeated behaviors 5-9 and behaviors 13-17. The number of times the loop is repeated is based on the social behaviors *SF7, SF8* and *SF9*.

Cycles in the execution chain prevent direct comparison between the performances of the teams in the looping groups to the others. To overcome this problem, we normalize the duration of the experiments in the looping groups by averaging their time spent in the cycles. In the following tables the performance times for groups 2, 6 were normalized. Experimental results for groups 8 and 9 are not presented due to the jump from behavior 1 to behavior 13.

5.3 Results and Conclusions

Since social behaviors that do not use connections have near zero performance time, our goal was to measure the time spent on communications between robots. We designed our experiments in such a way that at each decision point *A, B, C, D, E* various social behaviors were executed, which either did or did not use communications. **Table 3** presents time (in milliseconds) spent on communications between robots for executing social behaviors.

Num Group	Point A synchronization			Point C synchronization			Point D allocation		Point C synchronization			Point E allocation	
	SF 1	SF 8	SF5	SF 1	SF 7	SF 9	SF 2	SF3	SF 1	SF 4	SF5	SF2	SF6
1	0			0			0		0			0	
2	0					688	0		0			0	
3	0			0				403	0				433
4			1284	0			0		0			0	
5		663				624		544	0			0	
6	0				0			894		361			1002
7			1478	0			0				1422	0	

Table 3. Time measurements for executing social behaviors.
All data in this table is measured in ms. Data error of ±128 ms is possible.

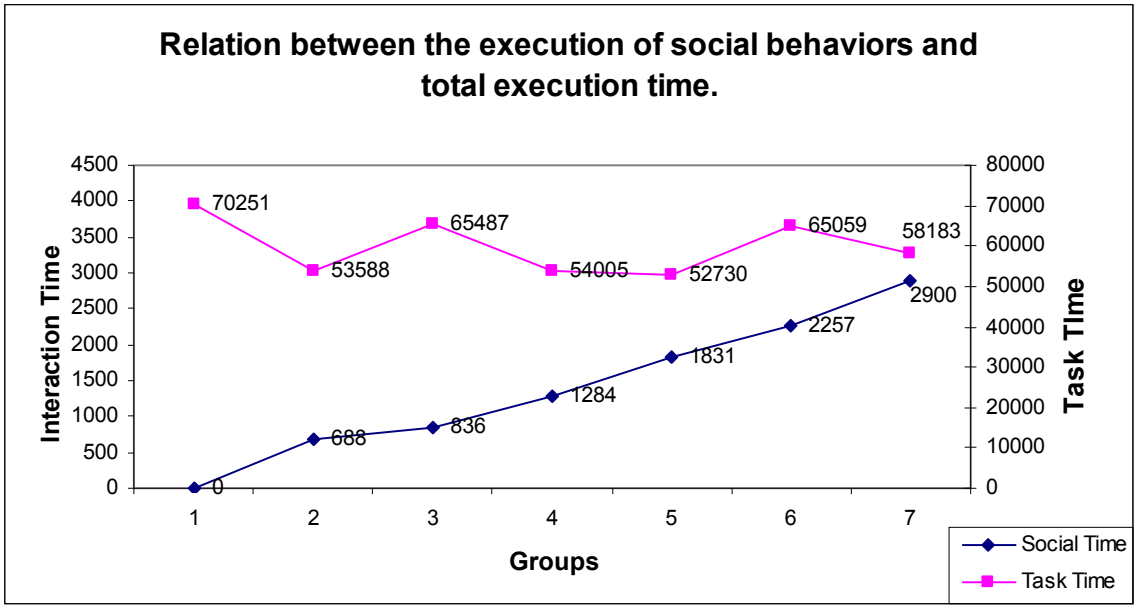
shows the average time spent on allocation and synchronization for each group experiment. It also presents the interaction time (sum of allocation and synchronization) and total average time of the experiments for each group. The data in the table is sorted by the average interaction time.

Group Number	Synchronization Time	Allocation Time	Interaction Time	Total Exper-iments Time
1	0	0	0	70251
2	688	0	688	53588
3	0	836	836	65487
4	1284	0	1284	54005
5	1287	544	1831	52730
6	361	1896	2257	65059
7	2900	0	2900	58183

Table 4. Average time spent on allocation and synchronization and total average time of the experiments for each group.

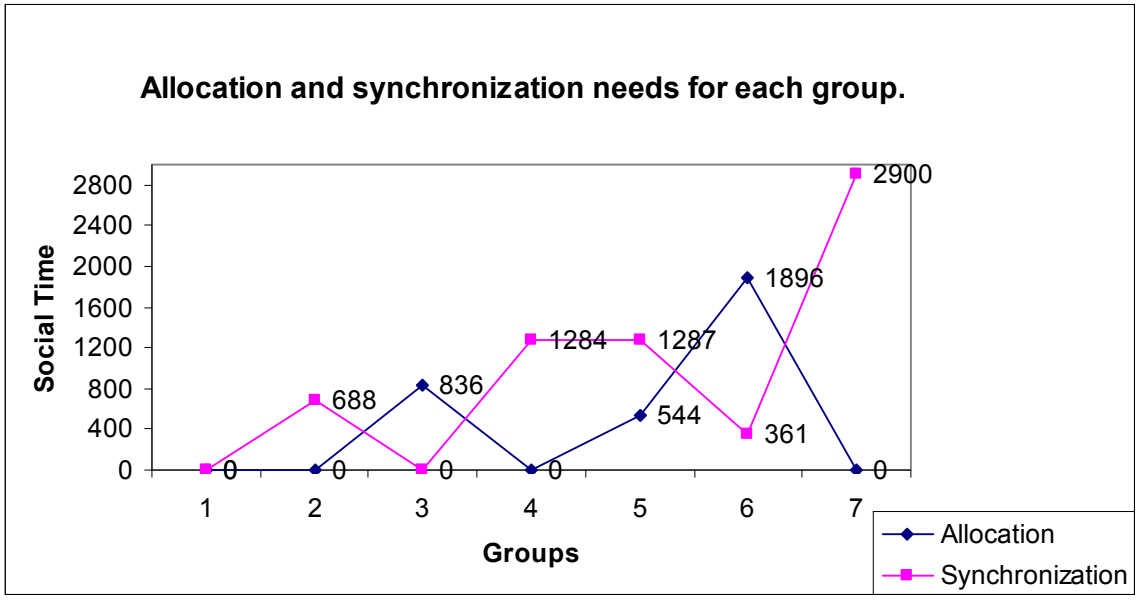
All data in this table are measured in ms.(Data error of $\pm$128 ms is possible.)

Graph 1 presents the relation between the time spent on execution of the social behaviors (*Synchronization time plus allocation time*) and total execution time of the experiments for each group. By creating various groups, we tried to combine different kinds of social behaviors to measure data on the wide execution time scale. This allowed us to analyze the influence of social interaction behaviors on the task execution time**. Graph 1** shows that increase in the social behavior execution time does not affect the total execution time.



Graph 1 . Relation between the execution time of social behaviors and total execution time (data from Table 4). Caption on both sides (Task Time, social interaction time)

Graph 2 shows deviations in time spent by each group on synchronization and allocation. It shows that groups spent different times for allocation and synchronization. Generally, groups that spent significant time on allocation spent much less time on synchronization and vice versa.



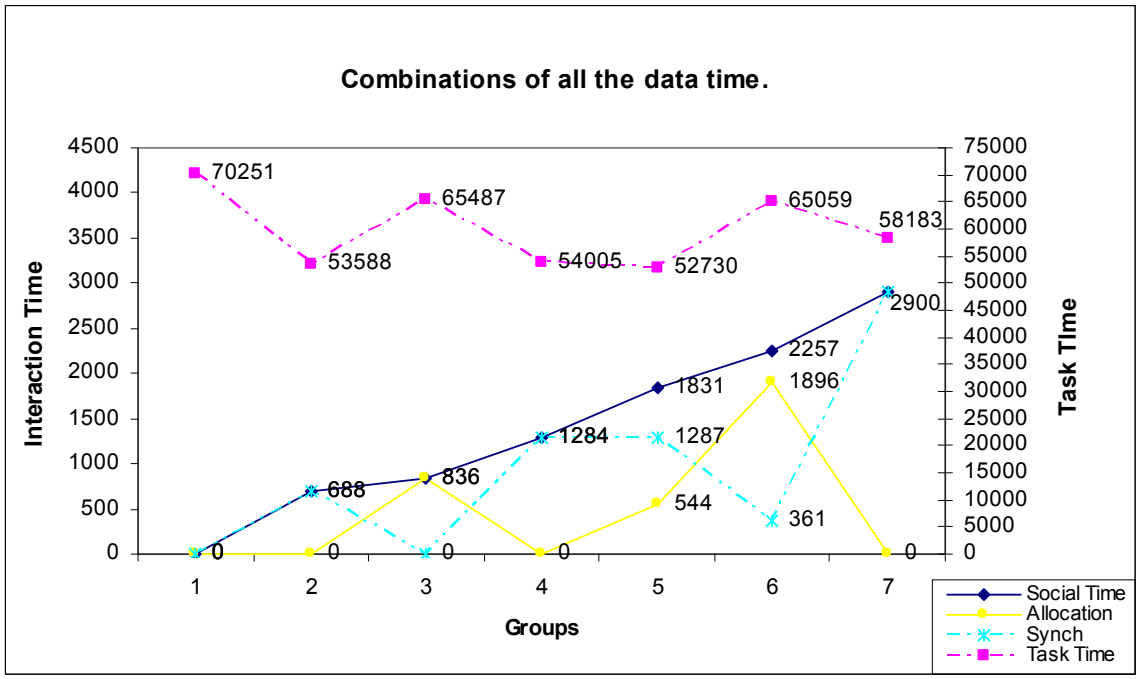
Graph 2. Allocation and synchronization needs for each group. (Table 4)

Axis X shows the names of the groups. Axis Y shows the time spent on social behavior for each group.

Graph 3a presents the combination of all measured time in the experiments for each group. We take special notice of time measured for groups 1, 3 and 6 (*subgroup A)* and groups 2, 4, 5 and 7 (*subgroup B*) presented separately in **Graph 3b** and **Graph 3c** respectively.

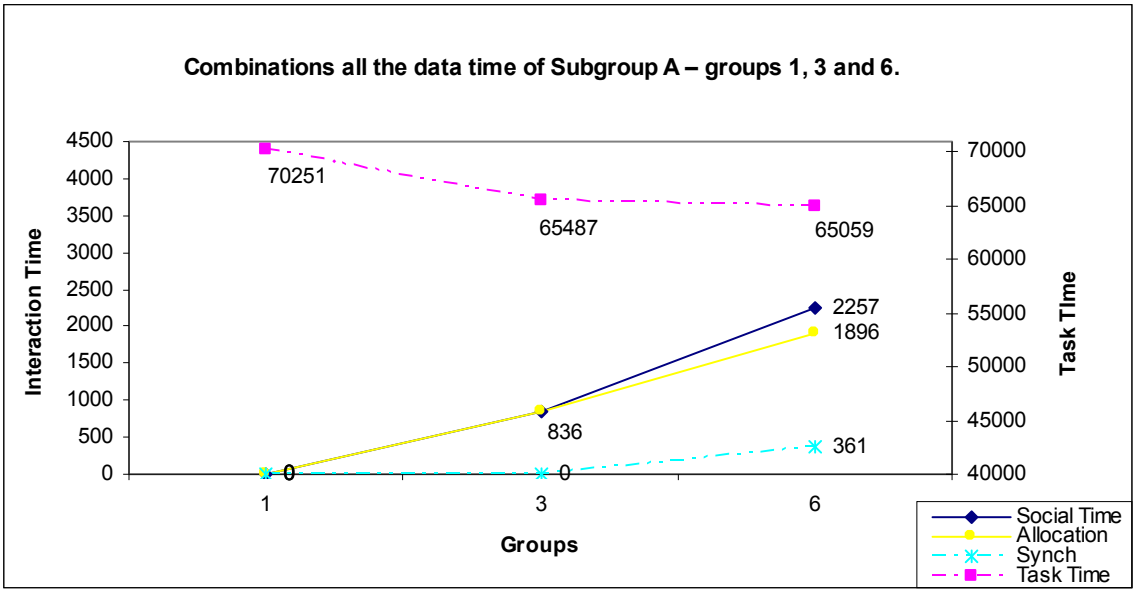
Each group in the *subgroup A* had a total execution time higher than the median (58183 ms). The social time, interaction time and allocation time increased, while total task time decreased. Group 6 had the lowest total execution time in the subgroup and the highest time spent on allocation and synchronization. It was the only group in *subgroup A* that used synchronization behaviors. From these results we can conclude that time spent on social

interaction and specifically on synchronization is beneficial to the architecture performance.

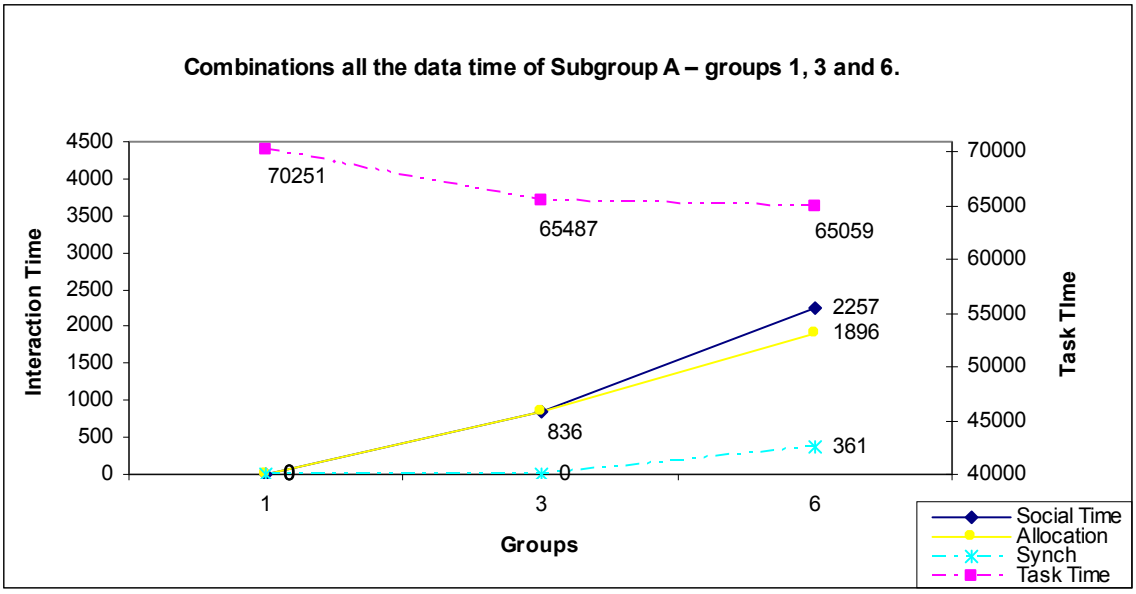


Graph 3. Combinations of all the data from Table 4.

The **Left** Axis Y shows the time spent on social behavior for each group the **Right** Y Axis shows the overall task time.



Graph 4. Combinations all the data time for *Subgroup A – groups 1, 3 and 6* (data from Table 4).



Graph 5c. Combinations all the data time for *Subgroup B – groups 2, 4, 5 and 7* (data from Table 4).

For *subgroup B* we notice the opposite trend. For most of the groups we notice the effect of social time: when social time increases, total execution time increases as well. The only group in the *subgroup B*, for which this is not true, is group 5 (that uses the allocation behavior). In light of this result we can conclude that time spent on allocation is also beneficial to architecture performance.

Only groups 5 and 6 used synchronization and allocation. Group 5 spent more time on synchronization while group 6 spent most of its time on allocation. From analyzing the total execution time for these groups, one can conclude that it is beneficial to spend more time on synchronization than on allocation. However, using only synchronization does not guarantee good results as can be seen from performance of the group 7.

We divided the results of our experiments into 3 policies: the first policy is where all the interaction time was spent on allocation, similarly the second policy is where all time was spent on synchronization; and the third policy is where interaction time included both allocation and synchronization. **Table 5** summarizes these results and shows that even for a simple behavior, the use of a variety of interaction behaviors makes a significant difference in task performance.

Policy	Synchronization Time	Allocation Time	Interaction Time	Task Time
Only Allocation	0	836	836	65487
Only Synchronization	1624	0	1624	55259
Synchronization&Allocation	824	1220	2044	58895

Table 5. Interaction and task times (ms).

Comparing the *Only Synchronization* and *Only Allocation* policies, we see that significant reduction in time spent on communication results in a 16% increase in total performance time. This shows that saving interaction time does not necessarily reduce task completion time. When synchronization and allocation were used together the interaction time increased noticeably. However, total task time was reduced by 10% comparing to the first policy (*Only Allocation)*, and increased only by 7% comparing to the second policy (*Only Synchronization*).

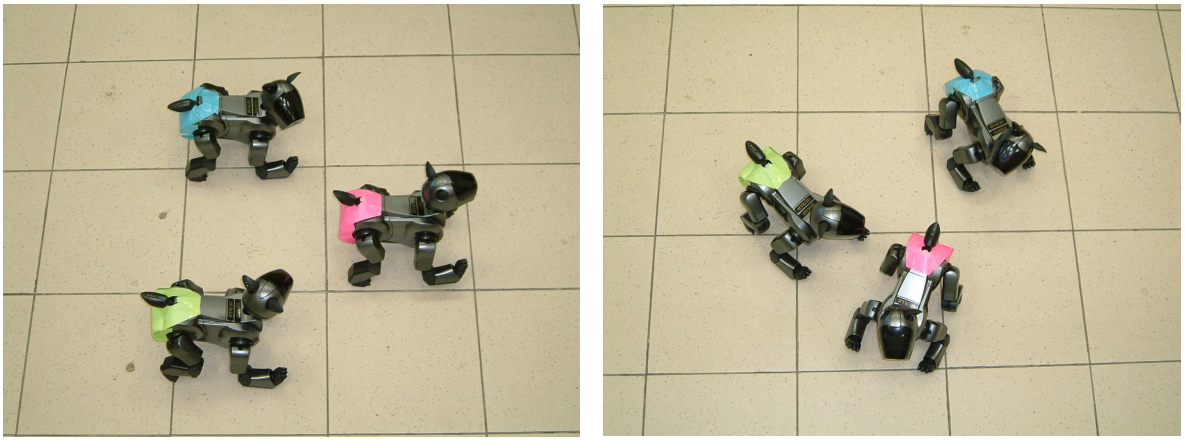
Therefore, our conclusion is that using both allocation and synchronization gives the desired results of the architecture. For different tasks it is possible to find the optimum set of social behaviors. BITE allows the designer to easily tune the architecture that will produce the best total execution time. Our results show that the ability to mix and match interaction behaviors can be a significant factor in task performance, signifying that integrating social interaction behaviors into the architecture is an important contribution.

5.4 Errors

During the execution of the experiments, we observed that at point *D* in the behavior graph the system was prone to errors.The errors occurred during execution of the social behavior *SF2* where robots had to decide on the role in the formation based on their ID. This social behavior does not check the position of the robots and the angle of their heads. Therefore, it can lead to a case where a robot on the left of the leader chooses to be the right follower and robot on the right chooses to be the left follower. In such cases, the two robots sometimes interfered with each other and therefore were unable to execute their tasks (Figure 12).

On the other hand, behavior SF3 assigns robots to their roles based on the angle of their head relative to the leader. Based on that the leftmost-robot is always assigned the task of the left-follower and the right-most robot is always assigned the task of the right-

follower. Therefore, executing behavior *SF3* at the allocation point *D* guarantees correct task assignment as was confirmed by our experiments.



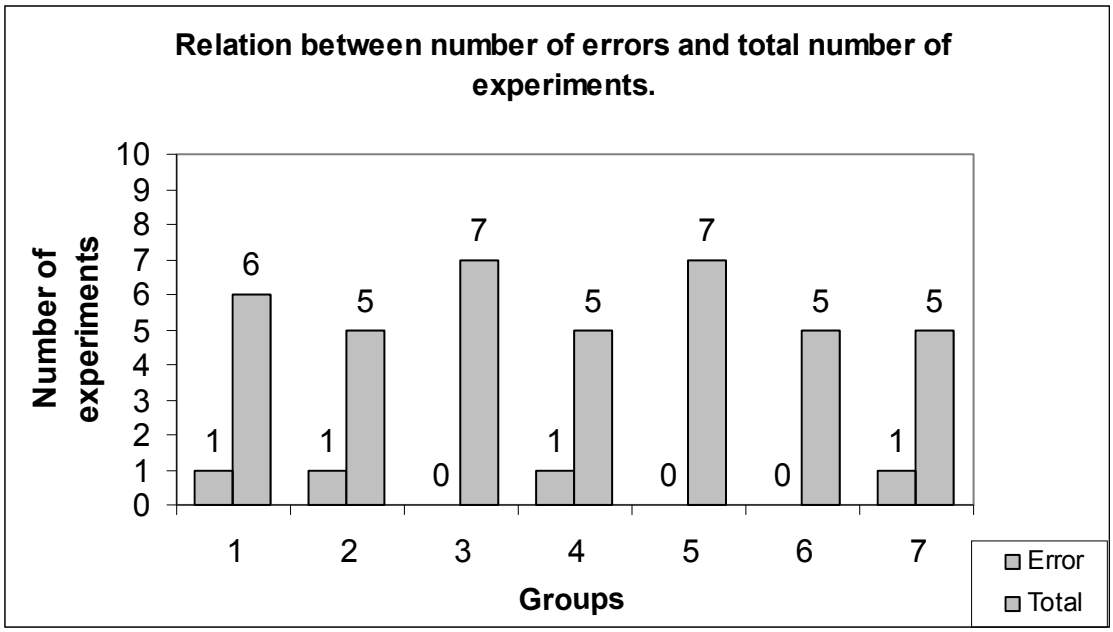
(a) Success **(b) Failure**

Figure 12. Robots executing triangle formation.

Graph 4 presents the number of erroneous cases occurred in each group configuration during the experiments. One solution to control such errors is to let the designer of the behaviors cover each specific erroneous case in the behavior itself. In our experiments we tried to prevent errors caused by the execution of the social behavior *SF2* by pre-positioning the robots prior to the experiments. However, we could not always succeed because this approach required us to anticipate every possible problem; therefore, 10% of all experiments were with errors.

However, BITE is inherently susceptible to sensing failures. For example, when a robot looses track of the leading robot in the Triangle Formation, an appropriate termination condition is satisfied and the *Walk* behavior terminates. This in turn triggers an appropriate social interaction behavior in BITE, which causes other robots to stop executing their *Walk* behavior, and jointly switch to the *Search* behavior.

To test the flexibility of our architecture, in some experiments we intentionally switched the robots, such that their roles in the formation would be reversed. The allocation interaction behavior automatically allowed the robots to re-select their role. Moreover, in cases where both robots were placed to the right of the leader, the allocation behavior decided which robot should remain as the right follower and which should be assigned as the left follower.



Graph 6. Relation between number of errors and total number of experiments.

5.5 Operator Control

It is important to realize that the integration of a human operator in the control loop of the robots is crucial. While the overarching goal is to achieve completely autonomous robots, most applications require a human to be involved in the decision making. Moreover, emphasis on autonomy is intended to allow a human operator to control several robots at once, rather than just one [2]. Thus teamwork architectures for physical robots must be designed to integrate a human operator.

Therefore, BITE allows an operator to control robots in several ways. *First,* the social interaction behaviors that manage synchronization and allocation may defer to a human. For instance, rather than executing a voting protocol, the synchronization interaction behavior may display a graphical window on the operator's console and consult the operator on which behavior the team should select. The operator is hidden behind the interaction behavior and thus robots continue running BITE while the operator makes a selection.

Second, BITE allows an operator to take control over a single robot without removing it from the behavior graph or disturbing the behavior execution stack (**Algorithm 4**). Thus the behaviors on the stack continue to be active and all interactions with the other robots (running BITE without the operator's control) continue to execute seamlessly. In addition, members under the operator control are automatically excluded from the usual decision-making process, but they do keep track of the execution and decisions of their team members. Therefore, when they are released, they ‘know’ what their team members are currently executing and what their future tasks will be.

Chapter 6

Summary and Future Work

A key challenge in building robot teams is to automate teamwork in such a way that the designer can focus on planning the robots' task work. Existing teamwork architectures do automate key aspects of teamwork, but the architectures are monolithic. They do not allow flexibility in achieving this automation in terms of the interaction protocols used. Existing architectures also do not allow mixing different protocols within the same overall robot team mission.

In this thesis we presented the BITE architecture that introduces a new key feature: the social interactions. It is implemented as behaviors which are managed through the same mechanisms as the task-oriented behaviors. This feature of BITE allows a significant degree of flexibility in teamwork design over existing architectures. This was the key motivation of our development.

BITE is a teamwork architecture based on distributed behaviors that enables the flexibility for behavior-based robot teams. It separates behaviors that control social interactions from those that manage subtasks. It enables creating of systems that combine the capabilities of different architectures and allow different interactions to be used depending on the context. The architecture of behaviors is very flexible and comfortable for constructing any other behavior graph for various purposes. The designer can easily change the type of social behavior in the definition of the behavior graph or he/she can build a new behavior graph to replace existing ones. Moreover, existing behaviors can be readdressed to different robot teams.

In this thesis we introduced new algorithms for controlling social interactions and communications within the BITE architecture. Our results from multiple experiments run on a team of Sony AIBO robots show that flexibility is very important and can significantly affect task performance.

Our future efforts focus on human-team interactions and on extending BITE's capabilities towards greater fault tolerance by integrating learning-algorithms. We also plan to investigate the use of BITE in multiple robotic platforms and to test its execution of a variety of other tasks on different robot platforms.

Bibliography

[1] M. B. Dias and A. T. Stentz. A free market architecture for distributed control of a multi-robot system. In *6th International Conference on Intelligent Autonomous Systems (IAS-6)*, pages 115.122, July 2000.

[2] Y. Elmaliach. Single operator control of coordinated robot teams. Master's thesis, Bar Ilan University, 2004.

[3] B. P. Gerkey and M. J. Mataric. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758.768, 10 2002. Special Issue on Multi-Robot Systems.

[4] 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.

[5] N. R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *AIJ*, 75(2):195.240, 1995.

[6] L. E. Parker. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220.240, April 1998.

[7] 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.

[8] M. Veloso, P. Stone, and M. 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.

[9] D. V.Pynadath and M. Tambe. Automated teamwork among heterogeneous software agents and humans. *AAMAS*, 7:71.100, 2003.

[10] 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.

[11] The Tekkotsu Homepage. Carnegie Mellon University : http://www-2.cs.cmu.edu/∨tekkotsu/, 2002–2003.

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

[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.

[14] Gerkey, Brian P. and Mataric, Maja J. 2003. Multi-robot task allocation: analyzing the complexity and optimality of key architectures IEEE international conference on robotics and automation (ICRA 2003).

[15] Sony Corporation: OPEN-R SDK Programmer's Guide (2004) http://aibo.com

[16] Kaminka, G. A. and Frenkel, I. 2005. Flexible Teamwork in Behavior-Based Robots. In Proceedings of the National Conference on Artificial Intelligence (AAAI-2005).

[17] Kaminka, G. A. and Frenkel, I. 2005. Towards Flexible Teamwork in Behavior-Based Robots: Extended Abstract. In Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-05).

[18] Kaminka, G. A., Elmaliach, Y., Frenkel, I., Glick, R., Kalech, M., Shpigelman, T. 2004. Towards a Comprehensive Framework for Teamwork in Behavior-Based Robots. In Proceedings of the Eighth Conference on Intelligent Autonomous Systems (IAS-8). IOS Press.

אוניברסיטת בר-אילן
מחלקה למדעי-המחשב

אינה פרנקל

עבודת צוות גמישה לרובוטים מבוססי התנהגות.

עבודה זו מוגשת כחלק מהדרישות לשם קבלת תואר מוסמך במחלקה למדעי המחשב של אוניברסיטת בר-אילן
רמת-גן, תשס"ה (2005)
עבודה זו נעשתה בהדרכתו
ד"ר גל קמינקא.