ADAPT: Abstraction Hierarchies to Better Simulate Teamwork [⋆]

Meirav Hadad[1] and Avi Rosenfeld[2]

[1] Research Division, Elbit Systems Ltd, Rosh Ha'Ayin 48091, Israel
[2] Jerusalem College of Technology, Jerusalem 91160, Israel
Meirav.Hadad@elbitsystems.com, rosenfa@jct.ac.il

Abstract. In this paper we present a lightweight teamwork implementation by using abstraction hierarchies. The basis of this implementation is ADAPT, which supports **A**utonomous **D**ynamic **A**gent **P**lanning for **T**eamwork. ADAPT's novelty stems from how it succinctly decomposes teamwork problems into two separate planners: a **task** network for the set of activities to be performed by a specific agent and a separate **group** network for addressing team organization factors. Because abstract search techniques are the basis for creating these two components, ADAPT agents are able to effectively address teamwork in dynamic environments without explicitly enumerating the entire set of possible team states. During run-time, ADAPT agents then expand the teamwork states that are necessary for task completion through an association algorithm to dynamically link its task and group planners. As a result, ADAPT uses far fewer team states than existing teamwork models. We describe how ADAPT was implemented within a commercial training and simulation application, and present evidence detailing its success in concisely and effectively modeling teamwork.

1 Introduction

Effectively quantifying teamwork problems is critical in many environments [4, 10]. However, one of the key challenges in creating teamwork models is how inter-agent rules can be encoded such that the team can still effectively behave in complex and dynamic environments [1, 10]. In particular, when multiple agents operate in these types of environments, their different mental states must be resolved so that a unified behavior can be formed for the team. One key research challenge for distributed artificial intelligence researchers is how these models can be created and implemented [10].

One leading solution is to decompose the group's actions into a set of rules which must be solved [8, 10]. Following this approach, the group's actions can be represented as a hierarchical structure of joint intentions and individual intentions and beliefs about others' intentions. However, this approach has two major drawbacks. First, the size of the model might be too large to realistically solve. Previous research found that many classes of teamwork problems exist for which finding the optimal sequence of actions is of intractable computational complexity [9]. Second, the structure of the tree must be flexible to dynamically changing conditions, such as changes in the environment, goal changes, and local or general constraints. Thus, even if a solution could be found

⋆ This research is based on work supported in part by Israel's Ministry of Science and Technology grant # 44115.

for a given time period, that solution might quickly become irrelevant. Hence, solutions must be found that reduce the size and structure of the team model such that it may be tractably and quickly solved, even in dynamic environments.

In this paper we present ADAPT, a novel approach for **A**utonomous **D**ynamic **A**gent **P**lanning for **T**eamwork. The key difference between ADAPT and other teamwork hierarchical approaches [1, 4, 5, 10, 11] stems from how teamwork is modeled. Previous approaches attempted to exhaustively depict all possible teamwork states. However, as has been previously demonstrated [9], the number of possible interactions between team members grows exponentially for many real-world domains, making these approaches difficult to implement, even in small to medium-sized groups.

Instead, ADAPT uses hierarchical abstraction as its basis in order to reduce the number of states which need to be considered. Specifically, a given teamwork problem is converted into two hierarchical networks: a **task** network to model the set of activities a given agent can perform and a separate **group** network for addressing organization factors. Within both hierarchical networks, behaviors are decomposed such that the general task and group problems are progressively redivided into partial plans involving smaller sets of subtasks and subgroups. ADAPT contains two novel elements designed to further reduce the size of these hierarchies. First, as hierarchical abstraction is used, agents incrementally add only relevant task and group information during task execution. Second, ADAPT uses an association algorithm to effectively perform task allocation. Agents only check those constraints which it may possibly perform, further adding to ADAPT's concise nature. The net result is that ADAPT can effectively simulate teamwork problems, even in dynamic environments, yet uses far fewer states than existing approaches.

While the ADAPT framework is general and is likely applicable to a variety of teamwork problems, in this paper we focus on how ADAPT was critical in implementing a multi-agent simulation. In Section 2 we present related teamwork models and compare those approaches to ADAPT, while Section 3 formally defines ADAPT and its algorithms. Sections 4 and 5 detail how ADAPT was implemented. Specifically, Section 4 focuses on describing the existing commercial multi-agent simulation into which ADAPT added. In Section 5 we discuss how ADAPT was successfully implemented into this framework, detail results which demonstrate the effectiveness of this framework in dynamic environments and show that the number of teamwork states that must be considered within ADAPT is significantly less than in other state-of-the-art approaches. This allowed the existing simulation to more effectively handle complex multi-agent tasks. Section 6 provides our conclusions.

2 Background and Motivation

Because of the importance of coordination problems, a variety of teamwork frameworks and formalizations have been proposed by the multi-agent research community [4, 1, 10]. The SharedPlans approach [1] consists of creating teamwork recipes based on modeling agents' beliefs and intentions. Tambe's STEAM teamwork engine [10] provides a set of generalized teamwork rules. The TAEMS framework [4] consists of hierarchical rule based approach where coordination relationships are quantified into groups, tasks, and methods.

ADAPT decomposes teamwork in a novel fashion by creating two hierarchical networks: a **task** network which addresses how the agent must plan its actions, and a **group** network that addresses how inter-agent assignments must be set. Previous work of multi-agent planning (e.g., [2]) and teamwork structures [4, 1, 10] suggested addressing the team's **task** planning as one multi-agent network which needs to be decomposed. Other works from social sciences [12] address how people within a team should be organized in order to facilitate the best planning of the activity. This approach parallels our creating a **group** network based on the agents in the team. However, ADAPT's novelty stems from applying abstract search techniques [7] to address multi-agent planning in its task and group network.

Previous approaches also separate team behavior into different components. Most similar to our approach, BITE is a behavior based teamwork architecture that separates task behaviors from behaviors between a single agent and its organization [5]. Similarly, ADAPT compartmentalizes teamwork between the task and the group. More generally, the TEAMCORE architecture uses a decision-theoretic structure to select different hierarchical team behaviors [11]. TAEMS separates team activities into tasks that are performed by the team with methods that can be performed by the agent [4]. However, in previous approaches, teamwork models were completely defined before task execution. They are required to explicitly define how every agent interacts with every other agent, and even how dynamics may affect these relationships, a process that can potentially lead to an exponential number of inter-agent states. When implementing these models, this state explosion can be prohibitively difficult as the number of team members grows.

In ADAPT, the task and group abstractions are incrementally built and dynamically changed during task execution. This difference allows us to significantly reduce the number of inter-agent states even when addressing dynamics. Additionally, ADAPT enables replanning for specific subproblems, allowing for more effective teamwork. Consequently, ADAPT allows for a more concise model which, in turn, facilitates easier simulation of complex, real-world tasks. We detail this approach in the next section.

3 Technique Description

ADAPT's model is based on taking a teamwork problem and then decomposing it into both task and group elaboration processes. As such, each of the task and group problems are decomposed in a top-down manner from a higher level, into progressively lower levels. The planning strategies of the elaboration processes in ADAPT are based on abstract search techniques [7]. Accordingly, the planning procedures of each elaboration process involves three major steps: (1) A *branching* step identifies possible candidates for expanding a partial plan; (2) A *refinement* step for adding constraint information to the partial plan; (3) a *pruning* step for removing unpromising candidates based on these constraints in order to avoid failure. While abstract-search is a well known technique for automated task planning [7], ADAPT's contribution stems from applying these techniques to teamwork modeling.

3.1 A Dynamic Planning Teamwork Example

To clarify how we intend to use these concepts, consider the following general example. Assume that a group must work as a team on a joint mission, say to capture a flag. A group of blue agents must plan how they will infiltrate the territory of the opposing team of red agents who are defending the flag. This type of scenario is typified in many real-world scenarios, such as military missions involving destroying an enemy target. In dynamic environments it is almost impossible to predict all possible event permutations that may occur while the blue agents complete their task.

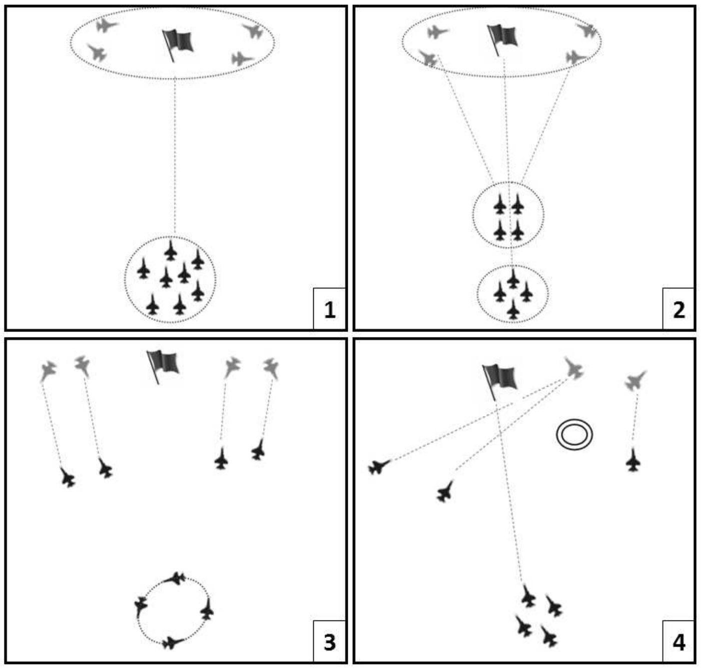

Fig. 1. Four Stages in a Mission Storyboard

Figure 1 depicts one series of group states during the execution of the “Capture the Flag mission”. At the start, a group of 4 red agents are divided into 2 subgroups of pairs located on either side of the flag to defend it (see the top left corner). At the same time, a group of 8 blue agents approach the flag area. In the second stage (see top right corner), the blue group splits into two subgroups of 4 agents according to their capabilities. One subgroup splits again into two subgroups of 2 agents and each subgroup approaches and engages the 2 red subgroups. In the next stage (bottom left) the blue agents engage the red ones to attempt to capture the flag. However, during this stage an unplanned event occurs, and one of the blue agents is incapacitated by a member of the opposing red team. The result of this change is that the group must replan their mission with only 7 of the 8 agents. In the final stage (bottom right), we see the group of 7 remaining blue agents still completing the task and capturing the flag.

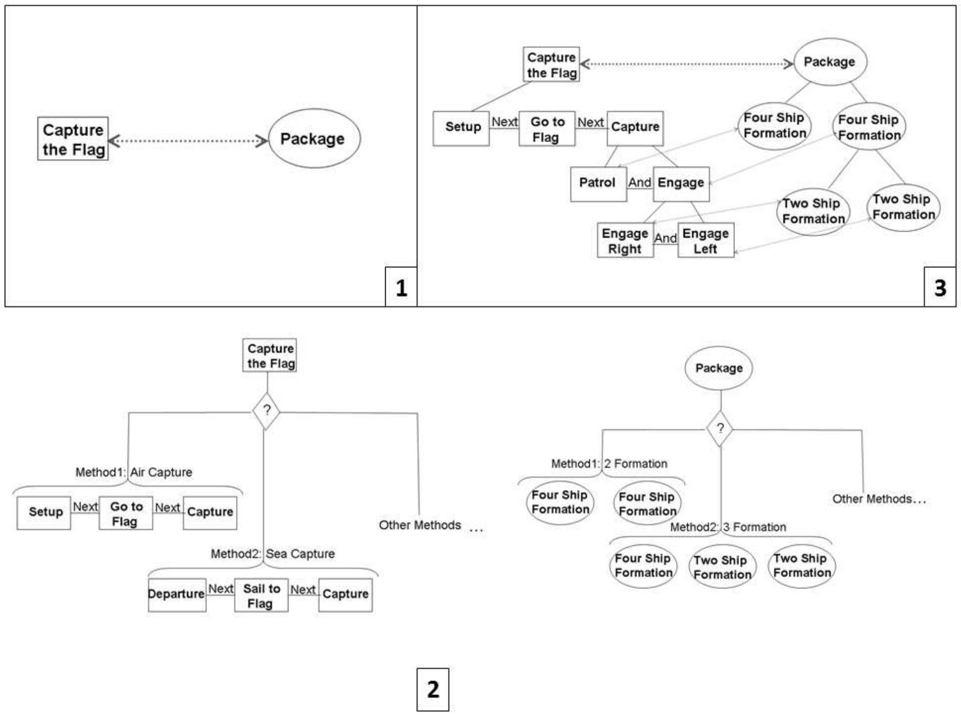

Fig. 2. Three Stages in Building the Teamwork Model in a Mission Storyboard (see Figure 1)

3.2 High Level Overview of ADAPT

While the ADAPT agents plan their task, they use the branching, refinement and pruning stages of abstract search techniques to limit the size of the teamwork model. We depict the stages of the teamwork model formation for the blue team in Figure 2. As previously described, ADAPT decomposes teamwork into both task and group networks. In the first stage (Stage 1 in Figure 2) each of these components are described only generally in the form of one abstract node. To graphically differentiate between the two task and group abstractions, we present the task hierarchy in rectangles, and the group hierarchy in ovals. At the beginning of execution, one rectangular task node describes the high level "Capture the Flag" task, and the group hierarchy "Package" describes the blue agents' attributes and capabilities which can be used to perform this task. In order for the blue agents to perform the team task, "Capture the Flag", their group and task planners must decide exactly how they will properly connect these two hierarchies. To make this decision, the agents' planners must apply their *branching* step to expand their abstract components of all applicable group and task options, which we refer to as *methods*. This is graphically represented in Stage 2 of Figure 2. However, unique to ADAPT and beyond similar previous teamwork approaches such as BITE and TEAMCORE [5, 10, 11], we then apply a *refinement* step where each agent generates the best applicable option based on its locally available information and the set of constraints associated with each option. We model each distributed agent as having a planner which uses a Distributed Constraint Optimization Problem (DCOP) solver to help create teamwork plans. In our implementation, the DCOP solver is based on the existing OptAPO algorithm [6]. As per the OptAPO algorithm, a mediator agent is elected which collects each of the distributed agents' constraints. In the next *punning* step, the mediator agent selects the best option given the choices each distributed agent presents. The mediator agent then informs each distributed agent about the option chosen, which is then selected by the local agent and executed.

Referring again to the example in Figure 2, the distributed planner decides that the best sequence for the blue agents to execute the team task, "Capture the Flag", is to first select the "Setup" subtask, then "Go to Flag", and lastly the "Capture" subtask. Within each subtask a further decomposition may occur into additional subtasks and subgroups. For example, the "Capture" subtask is decomposed into two subtasks which are assigned to 2 subgroups. One subgroup of four agents performs the "Patrol" subtask, while the second subgroup of four agents perform the "Engage" activity, where they engage the red agents defending the flag. The allocation step, where each agent is assigned to a given subtask, is also performed by the *refinement* step (Stage 3 of Figure 2). The best assignment is decided by the OptAPO mediator agent. As only a subset of all agents can perform certain activities, we can then apply the *pruning* step by which we reduce the teamwork model to only those states which are theoretically feasible. The mediator is also responsible for checking, or associating, between the task and group networks in order to ensure that the solution is feasible. Combining the *refinement* and *pruning* steps allows for a significantly smaller teamwork model than previous approaches [5, 10, 11] as their approaches stop model construction at the *branching* step. Thus, our work searches for a teamwork solution in a much smaller state space than in previous approaches.

In the following sections we formally describe and further detail the exact process by which these group and task networks are built. We also describe how these networks are associated such that teamwork problems can be solved in real-time and yet address dynamic changes from within the problem.

3.3 Modeling ADAPT's Constraint Networks

We model each task and group network as having a hierarchical structure which must be solved as a type of distributed constraint optimization problem (DCOP). Following previous DCOP work we define a DCOP problem as a set of variables where each variable is assigned to an agent who has control of its value. Cooperative agents must then coordinate their choice of values so that a global utility function is optimized. Formally, this process has previously been described as [6]:

- A set of N agents A = A_1, A_2 . . . , A_N
- A set of n variables V = X_1, X_2 . . . , X_n
- A set of domains D = D_1, D_2 . . . , D_n where the value of X_i is taken from D_i. Each D_i is assumed finite and discrete.
- A set of cost functions f = f_1, f_2 . . . , f_m where each f_i is a function f_i: $D_{i,1} \times \ldots \times D_{i,j} \rightarrow$ N $\cup\ \infty$. Cost functions are also called *constraints*.
- A distribution mapping Q : V $\rightarrow$ A assigning each variable to an agent. Q(X_i) = A_i denotes that A_i is responsible for choosing a value for X_i. A_i is given knowledge of X_i, D_i and all f_i involving X_i.

– An objective function F defined as an aggregation over the set of cost functions. Summation is typically used.

In the following sections we describe how we have implemented DCOP to create teamwork behavior in ADAPT's task and group network.

Modeling ADAPT's Task Network As our goal is to succinctly implement the simulation of group behavior, ADAPT contains many similarities to previous Hierarchical Task Network (HTN) planning approaches [4, 8, 7, 3] but includes extensions for dynamic multi-agent environments. Formally, we define an *atomic task* (or primitive task) as an action $act(\overrightarrow{v})$ that can be directly executed by the agents (e.g., $FlyTo(origin, dest)$). A (higher-level) *complex task* $c(\overrightarrow{v})$ is one that cannot be executed directly and is decomposed into subtasks (e.g., $Defend(v_1, v_2, v_3, v_4)$. Each task may be associated with two kinds of boolean formulas – a *precondition rule* and *postcondition rule* – to indicate the required situations for starting and ending the task execution (e.g., $(IsFuel > 200.lib) \wedge (IsTime =$ 5:00PM). We define tasks as being either a *single-agent task* or a *multi-agent task*. A single-agent task can be executed by one agent by itself and multi-agent tasks require 2 or more cooperative agents to complete the task.

To execute a high-level complex task $c(\overrightarrow{v})$, agents must identify a *method* that encodes all constraints for how this task may be performed, including key information about which agent can perform this task and constraints as to how it can be performed. Specifically, we define a method, m, as a 5-tuple containing:
$\langle$name(m), task(m), constr(m), subtasks(m), relation$(m)\rangle$, where name(m) is the name of the method and task(m) is the name of the complex task. We define $subtasks(m)$ as the sequence of tasks and $constr(m)$ as the set of constraints $\{\rho_1 \dots \rho_p\}$ that may apply when using the method m. Each constraint ρ_k involves a subset of variables and specifies all combinations of values for these variables. We define these variables as the set of $\{X_1 \dots X_n\}$ where each value X_i is taken from a set of D_i possible values for a given problem. Constraints may include specific required capabilities that a certain number of agents perform specific $subtasks(m)$. For example, there may be a constraint stating that the number of agents required to perform a subtask must be between 2 and 5 (formally, $2 \geq X_{agentNun} \leq 5$). Alternatively, these constraints may specify the type of agent that can perform a certain subtask, for example that the type of agent must be a fighter plane. In our implementation, we assumed these constraints were boolean. The relationship, $relation(m)$, contains constraints on the execution of the $subtasks(m)$ and may be one of the following: (i) AND denotes that the task(m) is accomplished iff all the $subtasks(m)$ are accomplished; (ii) OR denotes that the task(m) is accomplished iff at least one of the $subtasks(m)$ is accomplished; and (iii) NEXT orders constraints between $subtasks(m)$ such that one subtask must be performed before another. These constraints contain similarities to the QAF and NLE constraints within the TAEMS teamwork framework [4].

We define a *task network* $d_{task} = [G_{task}, \rho]$ as a collection of tasks that have to be accomplished under constraints ρ. The task network is represented by an acyclic digraph $G_{task} = (V_{task}, E_{task})$ in which V_{task} is node set, E_{task} is the edge set, and each node $v \in V_{task}$ contains a task. The *task planning domain* $\mathcal{D}_{task} = (\mathcal{M}_{task}, \mathcal{A})$

consists of a library task methods $\mathcal{M}_{task}$ of methods and library $\mathcal{A}$ of atomic tasks. A *task planning problem* is defined as a triple $P_{task} = \langle d_{task}, \mathcal{B}, \mathcal{D}_{task} \rangle$ where d_{task} is the task network to be executed, $\mathcal{B}$ is the initial state and $\mathcal{D}_{task}$ is the planning domain. A *task plan* is a sequence $act_1 \ldots act_n$ of atomic actions.

Given a task planning problem instance, the planning process involves the *branching*, *refinement* and *pruning* steps. The *branching* step is defined by retrieving the entire set of methods in $\mathcal{M}_{task}$ which may be applied to the required task. *Refinement* then has each local agent check its $constr(m)$ and send what it considers to be its best option to the mediator agent within the DCOP solver. More formally, given a set of possible applicable methods $\{m_1, \ldots, m_t\}$ each method contains constraints $constr(m_j)$ that contain sets of variables $\{X_1^{m_j} \ldots X_n^{m_j}\}$ where each value $X_i^{m_j}$ is taken from a set of $D_i^{m_j}$. Consistent to the general DCOP formalization, the ADAPT agent must minimize the cost functions $f = \{f_1, \ldots, f_m\}$ where each $f_i(d_{i,1}^{m_j}, \ldots, d_{i,k}^{m_j})$ is a function of $f_i : D_{i,1}^{m_j} \times \ldots \times D_{i,k}^{m_j} \rightarrow N \cup \infty$. The teamwork problem is considered solved if an assignment $A^* = \{d_1^{m_j}, \ldots, d_n^{m_j} | d_i^{m_j} \in D_i^{m_j}\}$ is found such that the global cost, F^{m_j}, is minimized. As DCOP problems have been proven to be NP-complete [6], keeping the search space as small as possible is critical for implementing a working application, especially one capable of running in real-time even as it handles dynamics.

In ADAPT's *pruning* stage, the mediator uses the OptAPO algorithm to search for this teamwork solution. If a solution for $\mathcal{M}_{task}$ cannot be constructed, the mediator agent asks each agent to iteratively select its next possible method until a solution is found. This process can either result with a plan being found, or a NULL plan in failure. Assuming dynamics change the environment, the entire planning process is repeated from the *branching* step.

For example, referring back to Figure 2, a complex task by the name of "Capture the Flag" is to be performed (Stage 1). The complex task may be decomposed according to a set of methods from $\mathcal{M}_{CapturetheFlag}$ which can be used to indicate different ways to plan this task (Stage 2). In this example, the selected method includes the subtask Setup which is an atomic task, while the subtask Capture is a complex subtask which must then continue to be decomposed by additional methods. Stage 3 in Figure 2 depicts the last stage in task network for $G_{CapturetheFlag}$.

Modeling ADAPT's Group Network In parallel to the task hierarchy, ADAPT also deconstructs teamwork into a group component to model constraints about which agents can perform given tasks. We refer to the hierarchy about the entities combined capabilities as the **group**. Parallel to our task definitions, we decompose the hierarchy as per the **group decomposition** into higher levels of **complex entities** and **atomic entities** which cannot be divided into further levels.

More formally, an *atomic entity* indicates a single agent and its basic capabilities $agent(\overrightarrow{v})$ (e.g., $Airplane(Engine, Fuel, \ldots)$). A (high-level) *complex entity* $c(\overrightarrow{v})$ indicates a multi-agent group that can be decomposed into subgroups. The decomposition of the *complex entity* into subgroups is done according to group decomposition *method*. Specifically, method m is defined as a 4-tuple:
$\langle \text{name}(m), \text{entity}(m), \text{constr}(m), \text{subgroups}(m) \rangle$, where name$(m)$ is the name of the *method* and entity(m) is the name of the *complex entity*. The $subgroups(m)$ indicates

either atomic or complex entities. Similar to task method the $constr(m)$ indicates set of constraints $\{\phi_1 \ldots \phi_r\}$ that may apply when using the method m. These constraints indicate the required capabilities from agents to be assigned to the $subgroups(m)$ and the different constraints on the group (e.g, maximum group members). A *group network* $d_{group} = [G_{group}, \phi]$ is a collection of groups that have been organized in a hierarchical manner under constraints ϕ. The group network is represented by $G_{group} = (V_{group}, E_{group})$ in which V_{group} is a node set, E_{group} is the edge set, and each node $v \in V_{group}$ contains group information.

The *group planning domain* $\mathcal{D}_{group} = (\mathcal{M}_{group}, \mathcal{E})$ consists of a library $\mathcal{M}_{group}$ of methods and a library $\mathcal{E}$ of atomic entities. A *group planning problem* is defined as a triple containing P_{group}, which is defined as $\langle d_{group}, \mathcal{B}, \mathcal{D}_{group} \rangle$ where d_{group} is the group network to be executed, $\mathcal{B}$ is a set of agents with their concrete capabilities and $\mathcal{D}_{group}$ is the planning domain. A *group plan* assigns agents to the appropriate nodes in the group network based on their capabilities in such a way that all the constraints are satisfied.

Similar to the task planning process, given a group planning problem instance, the planning process again involves the *branching*, *refinement* and *pruning* steps as well. The *branching* step is defined by retrieving the entire set of methods in $\mathcal{M}_{group}$ which may be applied to the *complex entity*. The *refinement* stage then has each local agent check its $constr(m)$ and send what it considers to be its best option to the mediator node within the DCOP solver. In the *pruning* stage the mediator node then checks all received constraints and checks if a solution for $\mathcal{M}_{group}$ can be constructed. If several solutions are possible, it selects the solution with the highest expected utility (or the lowest cost). If no plan can be formed based on these constraints, each agent iteratively selects its next possible method until a solution is found. This process can either result with a plan being found, or a NULL plan in failure. Assuming dynamics change the environment, the entire planning process is repeated from the *branching* step.

For example, referring back to Figure 2, a *complex entity*, “Package” contains all possible group configurations. The *complex entity* may be decomposed according to a set of methods from $\mathcal{M}_{Package}$ which can be used to indicate different group compositions to plan this group organization. In this example, the selected method “FourShip Formation” includes that a group decomposition of two groups of four agents to be formed from the *complex entity* “Package”.

3.4 Association to Create Teamwork

It is important to stress that after the *pruning* stages described above in both the task and group networks, the mediator agent must check the consistency, or what we refer to as the *association*, between these two sets of constraints to see what teamwork action should be selected. The association process serves as an intermediary between the DCOP mediators for both the task and group planners within the two abstract networks. The association process may connect one or more vertices of the task and group networks. Thus, the association enables loose coupling between the planners by allowing each of them to modify the corresponding plan independently.

The major steps of solving a teamwork problem are given in Algorithm 1. The teamwork problem is divided into a two separate networks, d_{task} and d_{group}. An initial network is represented as a single vertex of the highest level task or group. The group planner is responsible to assign the initial agent(s) to the vertex of the initial network and the association process creates a link between these initial networks (line 1). Then the task planner creates a partial plan by expanding its task network as much as possible based on the constraint's world state W currently available to the agents' mediator (line 2). These constraints will typically include data such as the current states of the environment (e.g. weather) or the informational status of the agent (e.g. fuel level or position). During the task planning process, the mediator is responsible to assign an agent (or agents) to subtasks in the task network. The mediator then sends a request with the proposed assignment to the association process so possible group constraints can be checked. The association process connects to the group mediator (line 3) which checks all possible ways a given task can be allocated by expanding its group network under the constraints of the task planner (line 4). This is our implementation of the branching step. The association process is then responsible for linking the new vertices that were added to the group network to the corresponding vertices in the task network (lines 6-7). ADAPT then applies the partial solution on the environment through interleaving planning with execution (line 8). In this way, plans are built incrementally during real-time. Note that steps 6–8 correspond to the *refinement* and *pruning* stages of abstract search techniques. Next, if it is impossible to generate a partial plan because of information obtained from the *refinement* step, the association sends a replanning request to either the task planner or group planner (lines 9-10), and each local agent sends additional constraints and the plan is expanded as described in the former section. Finally, the association algorithm checks if the changes to the available data cause conflicts with the existing assignments (lines 11-13). If any conflict with the existing plan is detected due to the dynamic changes to the environment, the entire process is repeated.

Algorithm 1 *The major steps for dynamic association*

```
Input: Initial vertices $v_t \in V_{task}$ and $v_g \in V_{group}$
Output: Teamwork plan for current world state W
01  Create initial links between $v_t$ and $v_g$;
02  while the task plan is not completed:
03    if request from task planner is received then:
04        Send request for elaboration to group planner
05        Receive the set of the vertices $V'_{task}$ and $V'_{group}$
06          if Can-Associate ($V'_{task}$, $V'_{group}$) then:
07            Generate-links ($V'_{task}$, $V'_{group}$)
08            Apply partial teamwork plan if possible;
09          else
10            Send request for replanning (task or group)
11     Receive perceptions from the world:
12       if the new data causes to conflicts between links
13         Send request for replanning (task or group)
```

4 Implementation Issues

We have implemented ADAPT within a commercial training and simulation system at Elbit Systems LTD. Elbit specializes in large-scale defense solutions in the areas of aviation, land and naval military systems with ten of thousands of workers worldwide. One division within Elbit has been developing sophisticated simulation systems such that personnel can be trained without the cost and potential risk of using actual equipment. Towards this goal, Elbit has already developed realistic simulators for airplane cockpits, naval stations and ground forces. We propose a new application that builds upon Elbit's existing simulators to simulate more complex team training missions through using ADAPT to help reduce the teamwork model size so it may be effectively implemented.

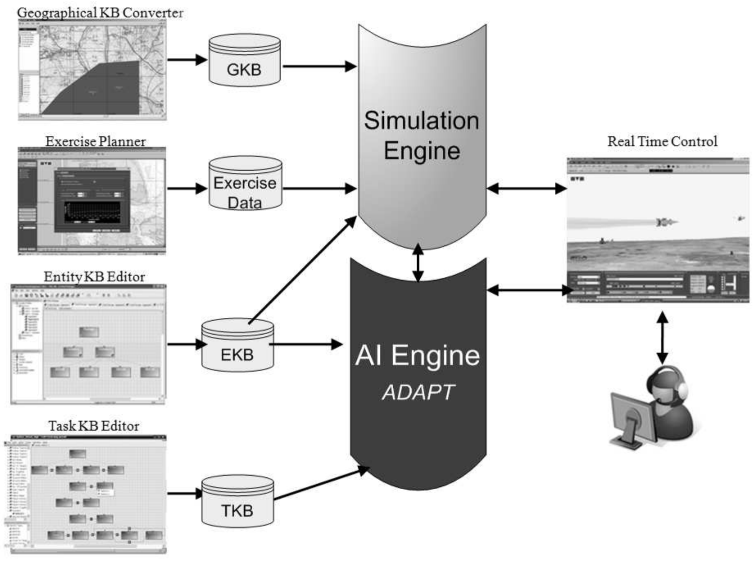

Fig. 3. A high level overview of the simulation system

Towards this goal, we created a working system at Elbit by integrating ADAPT within existing single-workstation simulation systems. Figure 3 depicts a high level description of the system's four major components. Elbit's previously developed simulation engine is still responsible for creating the base simulation environment. As part of this component, a Geographical Knowledge Base (GKB) contains geographical data about the training scenario and an exercise planner (EP) database is created with initial data of the training exercise (e.g., agents types, agents' forces, their initial location, their initial mission). Special to the ADAPT project, a Entity Knowledge Base (EKB) is created containing properties on each agent (e.g. aircraft type, max, min velocity). In addition, it includes various types of entities, including complex entities (e.g., platoon, battalion), and their decomposition methods that describes possible ways of decomposing the groups into subgroups. Thus, this database contains all relevant information about ADAPT's group network. Also, a Task Knowledge Base (TKB) is created containing a set of tasks that the agent can perform and their appropriate methods. Within military applications, this database represents a **doctrine**, or the key task that must be performed, or ADAPT's task network. Agents' decisions are based on the dynamic and static knowledge that the agents gather from the simulation engine as well as the constraint information in the EKB and the TKB. The Real Time (RT) control component enables the human trainer to interact with the simulated arena. Additionally, it provides the human interface to the simulation system.

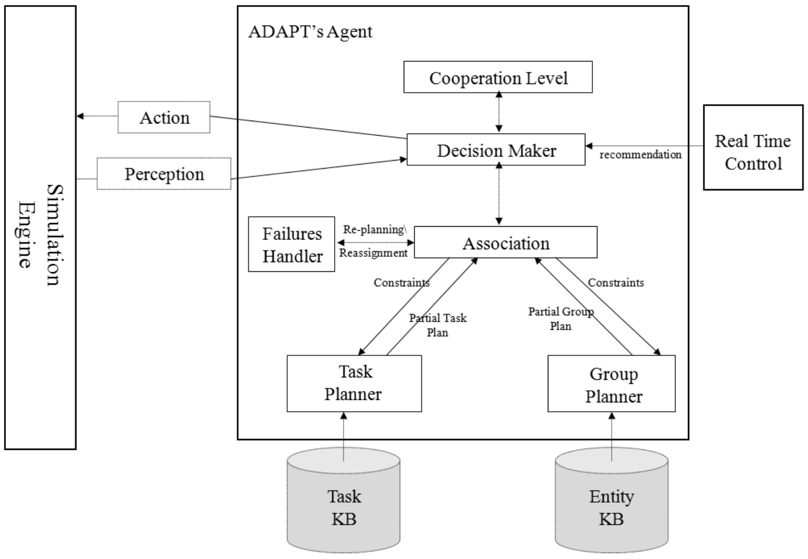

Fig. 4. General description of an ADAPT agent

Figure 4 provides a detailed description of how ADAPT and the algorithms presented in section 3, are integrated as the basis for creating this behavior. Moving from top to bottom within the Figure, each simulated entity is comprised of: a decision making process; a cooperation level; a failure handling process; and two types of planners (connected through the association process). Note that the Task Planner in Figure 2 is used to solve ADAPT's task network, and the Group Planner is used to solve ADAPT's group network. The decision maker is responsible for receiving the agent's perceptions and deciding on the agent's next steps accordingly.

5 ADAPT's Usefulness in a Simulation System

In studying ADAPT's usefulness in Elbit's simulation system, we focused on three key issues: 1. Can content experts easily work with the application to effectively impart their knowledge? 2. Does ADAPT indeed succinctly model teamwork, and how does it compare with other state of the art models? 3. Does the system perform effectively, and can it deal with system dynamics?

Specifically, we applied the general technique in Section 3 regarding the Capture the Flag problem, and applied this technique to scenarios involving fighter jets attempting to destroy an enemy target. Each scenario involved a target that needed to be destroyed, as well as groups of attacking and defending planes. The attacking planes form the blue group and are constructed from bomber and fighter planes (e.g. F16 fighters and Stealth bombers), and the defending group consist exclusively of red fighter planes (F16). The goal of the blue fighters is to disable the enemy's red fighters after which the blue bombers are able to destroy the target. The scenarios focused on different group sizes for the blue and red teams. Dynamics focus on unknown issues including the number of planes on each team that were disabled. In order to create the task and group networks, we consulted with a group of professional fighter pilots whose expert knowledge was then directly encoded. We relied on these experts to provide details about how they would perform theoretical missions. We then successfully encapsulated this information as the Doctrine and Technical databases to form ADAPT's task and group methods. A pictorial description of one scenario involving seven blue and six red planes is given in Figure 5.

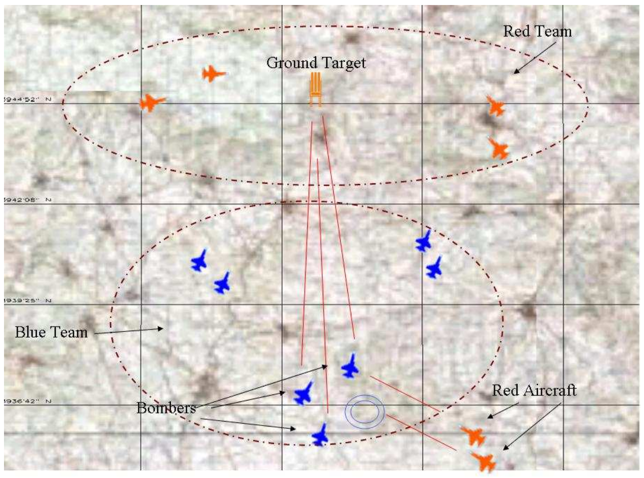

Fig. 5. Attack a ground target - simulation view snapshot

To study the savings in the number of states within ADAPT versus other previous static approaches [5, 10, 11], we focused on missions with groups of 5, 8 and 12 blue planes which needed to destroy one target on the red team guarded by a fixed number of 5 jets. We recorded the number of task and group nodes required to encode teamwork within ADAPT throughout the task's execution. We then compared how many states would be needed in these same problems by BITE [5]. We decided to compare the number of states needed by BITE as it too divides teamwork into Task and Group hierarchies and thus is the closest comparable model to ADAPT. However, as ADAPT uses abstract search as well, we would expect BITE to use a fixed number of possible task and group permutations, while ADAPT would only store those states actually needed to deal with problem execution. Furthermore, one would expect that the number of states in ADAPT can and will change during task execution, especially as problem dynamics are addressed. To study this point, we assumed 2 blue agents were disabled during task execution.

	BITE		**ADAPT** max		**ADAPT** average	
Number of Agents	Task States	Group States	Task States	Group States	Task States	Group States
5	561	18	44	5	37.1	3.67
8	624	146	53	8	39.65	6.29
12	829	400	68	8	56.86	6.17

Table 1. Comparing the number of task and group teamwork states in ADAPT versus BITE teamwork models

As Table 1 demonstrates, we found that ADAPT's use of abstraction yielded an enormous savings in the number of teamwork states needing to be stored. In columns 2 and 3, we present the size of BITE's task and group network within the problems we implemented. Compare these values to the **maximal** size of ADAPT's task and group networks in columns 4 and 5. The average state size is even smaller, and is presented in columns 6 and 7. These very significant savings are because ADAPT only stores task and group network nodes that are found to be relevant based on the current conditions as dictated by *branching*, *refinement*, and *pruning* stages. In contrast, static approaches such as BITE must preplan for all possible contingencies. This difference becomes more pronounced as ADAPT uses real-time planning based on the agent's current state. ADAPT interleaves planning and execution and thus applies partial group and task networks. This is why ADAPT has no need to create complete plans for all contingencies in advance. The net result is that ADAPT's group and task networks are initially defined abstractly and incompletely and built incrementally only as needed, based on the specific environment settings that the agents encounter during task execution based on ADAPT's associative algorithm. Thus, the maximum number of task and group nodes within ADAPT is far larger than its average. This difference can be observed by comparing the differences in the maximal model size and the average size for task states (columns 4 and 6) and group states (columns 5 and 7).

In addition to studying the size of ADAPT's teamwork model, we also evaluated the ease by which ADAPT could be implemented to verify that in fact it did facilitate tractably computing the team's optimal behavior even when faced with dynamics. Recall that the task and group planners are based on a state-of-the-art DCOP algorithm [6] to solve these constraints. However, as these problems are NP-complete, no DCOP algorithm can yield definite performance guarantees for all theoretical problems. As our production simulation must be able to run without noticeable lags, even when simulating complicated scenarios with high levels of dynamics, we believe that having a

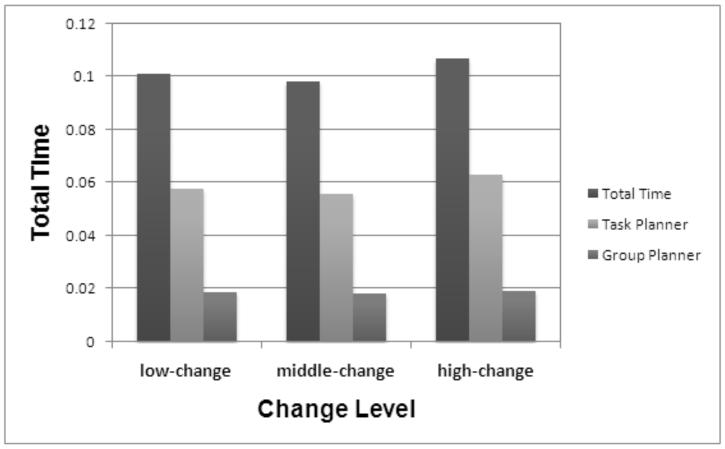

Fig. 6. The influence of dynamics on ADAPT's teamwork planners.

smaller teamwork model is critical towards achieving this goal. To evaluate this point, we implemented 3 variations of scenarios involving a team of 8 blue agents attempting to achieve their joint mission, i.e. *attack ground target*, versus a group of 5 red agents. To study the impact of dynamics on ADAPT, three levels of dynamic changes were tested: low-change, middle-change and high-change. In the low-change scenario the red force tried to defend the ground target but could not eliminate any of the blue force members and the group planner did not need to replan due to dynamics. This case represents the baseline of the study, as it allowed the blue force to complete its task with no changes in its force and with little need to change its mission plans. In the middle-change scenario, the red force succeeded in eliminating one or two of the blue fighters from the arena (based on non-deterministic effects), triggering some changes in the group hierarchy of fighters and requiring a moderate degree of mission and group replanning. In the high-change scenario the red force succeeded in eliminating three or more planes from both the fighter and bomber planes, causing more changes in the group hierarchy. This necessitated significant replanning efforts in both the task and group networks.

We measured the total planning time needed by the blue team agents using ADAPT to plan successful joint missions. We defined mission success as the elimination of the ground target and the blue team returning home. To examine ADAPT's performance, we compared the time needed by its problem solvers in 30 trials for each of the 3 different levels of dynamic changes (90 total trials) from sets of 5 minute simulations. We ran the ADAPT simulation on a 2.8 GHz Pentium D computer with 2 GHz of memory.

Figure 6 shows the total time utilized by the task planner, group planner and the decision maker to completely plan the joint mission. The total time represents the overall time used by the ADAPT engine to solve the teamwork problem. This time includes the component needed by the task planner, the group planner and the association process. In all cases, the task and group planners operated within fractions of seconds. Similarly, the total time used by ADAPT was under 0.12 seconds in even the most dynamic scenarios. Thus, we found that ADAPT facilitated real-time teamwork simulation, even in highly dynamic environments.

6 Conclusions

In this paper, we present ADAPT, a framework to decompose teamwork into abstract task and group networks. As ADAPT is the first teamwork model to use abstract search methods, it represents a radical departure over previous models which need to exhaustively describe all possible interactions prior to task completion [5, 10, 11]. As these models can be of exponential size, the problem of finding the optimal teamwork behavior can be of intractable complexity [9]. In contrast, ADAPT builds teamwork models incrementally during task execution, thus allowing agents to apply *refinement* and *pruning* steps to limit the size of the teamwork model needing to be stored. This fundamental difference not only yields teamwork models that are smaller by several orders of magnitude, but allows agents to quickly find their optimal behavior within this smaller model as described in this paper.

This paper also described how ADAPT was implemented within a challenging military simulation domain. We present results pertaining to how ADAPT formed the basis of a commercial system. We detail the specific task and group networks ADAPT created, how ADAPT can handle domain dynamics, and the time required by ADAPT to identify the optimal team behavior. While we have only implemented ADAPT to date in one series of planning problems, we are confident that this approach can be equally successful in other planning and scheduling problems due to ADAPT's generality.

References

1. B. J. Grosz and S. Kraus. Collaborative plans for complex group action. *AIJ*, 86(2):269–357, 1996.
2. M. Hadad, S. Kraus, Y. Gal, and R. Lin. Time reasoning for a collaborative planning agent in a dynamic environment. *Annals of Math. and AI*, 37(4):331–380, 2003.
3. H. Hoang, S. Lee-Urban, and H. Muoz-Avila. Hierarchical plan representations for encoding strategic game AI. In *AIIDE-05*. AAAI Press, 2005.
4. B. Horling, V. Lesser, R. Vincent, T. Wagner, A. Raja, S. Zhang, K. Decker, and A. Garvey. The TAEMS White Paper, January 1999.
5. G. A. Kaminka and I. Frenkel. Integration of coordination mechanisms in the BITE multi-robot architecture. *ICRA-07*, pages 2859–2866, 2007.
6. R. Mailler and V. Lesser. Using Cooperative Mediation to Solve Distributed Constraint Satisfaction Problems. In *AAMAS '04*, pages 446–453, New York, 2004.
7. D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
8. D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. Shop2: An HTN planning system. *J. Artif. Intell. Res. (JAIR)*, 20:379–404, 2003.
9. D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *JAIR*, 16:389–423, 2002.
10. M. Tambe. Toward flexible teamwork. *JAIR*, 7:83–124, 1997.
11. M. Tambe, D. V. Pynadath, N. Chauvat, A. Das, and G. A. Kaminka. Adaptive agent integration architectures for heterogeneous team members. pages 301–308, 2000.
12. R. W. Toseland and R. F. Rivas. *An Introduction to Group Work Practice*. Allyn and Bacon, Boston, 2001.