Evolution of a Teamwork Model

Nathan Schurr, Steven Okamoto, Rajiv T. Maheswaran,
Paul Scerri and Milind Tambe

1 Introduction

For heterogeneous agents working together to achieve complex goals, teamwork (Jennings, 1995; Yen, Yin, Ioerger, Miller, Xu, & Volz, 2001; Tambe, 1997a) has emerged as the dominant coordination paradigm. For domains as diverse as rescue response, military, space, sports and collaboration between human workmates, flexible, dynamic coordination between cooperative agents needs to be achieved despite complex, uncertain, and hostile environments. There is now emerging consensus in the multiagent arena that for flexible teamwork among agents, each team member is provided with an explicit model of teamwork, which entails their commitments and responsibilities as a team member. This explicit modelling allows the coordination to be robust, despite individual failures and unpredictably changing environments.

Building on the well developed theory of joint intentions (Cohen & Levesque, 1991) and shared plans (Grosz & Kraus, 1996), the STEAM teamwork model (Tambe, 1997a) was operationalized as a set of domain independent rules that describe how teams should work together. This domain independent teamwork model has been successfully applied to a variety of domains. From combat

air missions (Hill, Chen, Gratch, Rosenbloom, & Tambe, 1997) to robot soccer (Kitano, Asada, Kuniyoshi, Noda, Osawa, & Matsubara, 1997) to teams supporting human organizations (Pynadath & Tambe, 2003) to rescue response (Scerri, Pynadath, Johnson, P., Schurr, Si, & Tambe, 2003), applying the same set of STEAM rules has resulted in successful coordination between heterogeneous agents. The successful use of the same teamwork model in a wide variety of diverse domains provides compelling evidence that it is the principles of teamwork, rather than exploitation of specific domain phenomena, that underlies the success of teamwork based approaches.

Since the same rules can be successfully used in a range of domains, it is desirable to build a reusable software package that encapsulates those rules in order to provide a lightweight and portable implementation. The emerging standard for deploying such a package is via proxies (Pynadath & Tambe, 2003). Each proxy works closely with a single domain agent, representing that agent in the team. The second generation of teamwork proxies, called Machinetta (Pynadath & Tambe, 2003; Scerri et al., 2003), is currently being developed. The Machinetta proxies use less computing resources and are more flexible than the proxies they have superseded.

While approaches to teamwork have been shown to be effective for agent teams, new emerging domains of teamwork require agent-human interactions in teams. These emerging domains and the teams that are being developed for them introduce a new set of issues and obstacles. Two algorithms that need to be revised in particular for these complex domains are the algorithms for adjustable autonomy (for agent-human interaction) and algorithms for role allocation. This chapter focuses in particular on the challenge of role allocation.

Upon instantiation of a new plan, the roles needed to perform that plan are created and must be allocated to members of the team. In order to allocate a

dynamically changing set of roles to team members, previous mechanisms required too much computation and/or communication and did not handle rapidly changing situations well for teams with many members. A novel algorithm has been created for role allocation in these extreme teams. Generally in teamwork, role allocation is the problem of assigning roles to agents so as to maximize overall team utility (Nair, Ito, Tambe, & Marsella, 2002; Tidhar, Rao, & Sonenberg, 1996; Werger & Mataric, 2000). Extreme teams emphasize key additional properties in role allocation: (i) domain dynamics may cause tasks to disappear; (ii) agents may perform one or more roles, but within resource limits; (iii) many agents can fulfill the same role. This role allocation challenge in extreme teams will be referred to as extended GAP (E-GAP), as it subsumes the generalized assignment problem (GAP), which is NP-complete (Shmoys & Tardos, 1993).

2 Before Machinetta: STEAM in Soar

Machinetta has evolved from STEAM, which was implemented in Soar (Newell, 1990), and thus has historical roots in the Soar language. For more on Soar, refer to Chapter 3 of this volume. The two aspects of Machinetta where Soar's influence is most apparent are the Team Oriented Plan (TOP) and the coordination component (see Section 3).

Even though there has been a conversion to Machinetta, the team plans come from Soar. The language and syntax used to describe the TOP are derived from the syntax of the operators that Soar used. This allows for the same human readable expression of team plans that STEAM had.

The Belief-Desire-Intention (BDI) (Georgeff, Pell, Pollack, Tambe, & Wooldrige, 1998) framework of joint intentions (Cohen & Levesque, 1991) is used to guide communication between proxies. This takes the form of a policy which decides which beliefs to communicate and which proxies to communicate these beliefs

to. For Machinetta, these policy algorithms were translated from Soar into Java in the coordination component of the proxy. For an example of some of these Soar rules, see Appendix A.

Indeed, the Soar model can be viewed as a BDI architecture, enabling us to borrow from BDI theories. In the rest of this section, a mapping of Soar to BDI is presented, and readers unfamiliar with Soar may wish to proceed forward to Section 3.

To see the mapping from Soar to BDI, let us consider a very abstract definition of the Soar model. Soar is based on operators, which are similar to reactive plans, and states (which include the agent's highest-level goals and beliefs about its environment). Operators are qualified by preconditions which help select operators for execution based on an agent's current state. Selecting high-level operators for execution leads to subgoals and thus a hierarchical expansion of operators ensues. Selected operators are reconsidered if their termination conditions match the state. While this abstract description ignores significant aspects of the Soar architecture, such as (i) its meta-level reasoning layer, and (ii) its highly optimized rule-based implementation layer, it will be sufficient for the sake of defining an abstract mapping between BDI architectures and Soar as follows:

1: Intentions are selected operators in Soar

2: Beliefs are included in the current state in Soar

3: Desires are goals (including those generated from operators which are subgoals)

4: Commitment strategies are strategies for defining operator termination conditions. For instance, operators may be terminated only if they are achieved, unachievable or irrelevant

In Soar, a selected operator (commitment) constrains the new operators (options) that the agent is willing to consider. In particular, the operator constrains the problem space that is selected in its subgoal. This problem space in turn constrains the choice of new operators that are considered in the subgoal (unless a new situation causes the higher-level operator itself to be reconsidered). Interestingly, such insights from Soar have parallels in BDI architectures. Both Soar and BDI architectures have by now been applied to several large-scale applications. Thus, they share concerns of efficiency, real-time, and scalability to large scale applications. Interestingly, even the application domains have also overlapped. For instance, PRS and dMARS have been applied in air-combat simulation, which is also one of the large-scale applications for Soar.

Despite such commonality, there are some key differences between Soar and conventional BDI models. Interestingly, in these differences, the two models appear to complement each other's strengths. For instance, Soar research has typically appealed to cognitive psychology and practical applications for rationalizing design decisions. In contrast, BDI architectures have appealed to logic and philosophy. Furthermore, Soar has often taken an empirical approach to architecture design, where systems are first constructed and some of the underlying principles are understood via such constructed systems. Thus, Soar includes modules such as chunking, a form of explanation-based learning, and a truth maintenance system for maintaining state consistency, which as yet appear to be absent from BDI systems. In contrast, the approach in BDI systems appears is to first clearly understand the logical and philosophical underpinnings and then build systems.

3 Machinetta Proxies

Proxies are pieces of software that facilitate the actions and communication necessary for robots, agents and people (RAPs) to work cooperatively on a team plan. Each team member has a proxy that represents it in team collaboration. This section will describe the inner workings of a Machinetta proxy. Machinetta proxies are implemented as lightweight, domain-independent Java programs, capable of performing the activities required to get a large group heterogeneous entities to work together. The proxies are designed to run on a number of platforms including laptops, robots and handheld devices.

3.1 Components

The Machinetta proxy's software is made up of five components as seen in Figure 1. Each component abstracts away details allowing other components to work without considering those details.

Communication: communication with other proxies

Coordination: reasoning about team plans and communication

State: the working memory of the proxy

Adjustable Autonomy: reasoning about whether to act autonomously or pass control to the team member

RAP Interface: communication with the team member

The adjustable autonomy component addresses the circumstances under which the proxy should act autonomously as opposed to waiting for input from a team member. Such reasoning is vital to the successful deployment of heterogeneous teams containing people. However, other components and proxies are insulated from this reasoning process by the adjustable autonomy component

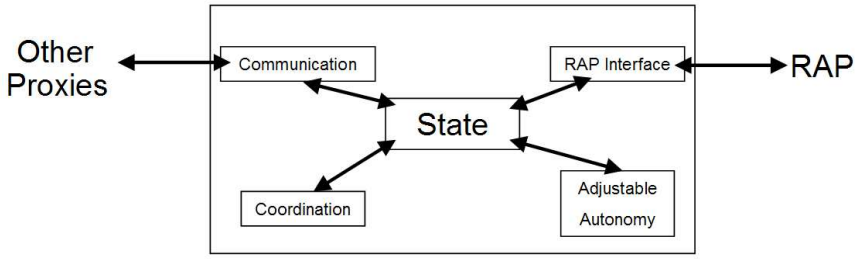


Figure 1: Proxy software architecture.

and need only know the ultimate decision made by the proxy, whether that decision was made autonomously or by the team member.

The RAP interface component is the only part of the proxy that needs to be designed for a specific type of team member. For example, the RAP interface for a person playing the role of fire chief in the disaster rescue domain is a large graphical interface, while for agents a simple socket communicating a small, fixed set of messages is sufficient. With some extensions, these techniques were used to allow Machinetta to scale up to run 200 proxies on two desktop computers.

3.2 TOP

A team of proxies implements Team Oriented Plans (TOPs). A TOP is a team-level description of the activities that need to be performed in order to achieve the goals of the team. It consists of reactive team plans, roles, relationships between roles, and conditions for initiating a plan and terminating a plan. The proxies dynamically instantiate plans when, during the course of execution, their current states match a plan's required trigger conditions. The proxy communication policy determines precisely which messages should be sent among proxies to ensure that cohesion is maintained.

In developing Machinetta, much of the focus has been on joint intentions

theory (Cohen & Levesque, 1991) due to its detailed formal specification and prescriptive power. The joint intentions framework provides a modal logic specification of a team's mental state, called a joint intention. A team has a joint intention to commit a team action if its team members are jointly committed to completing that team action, while mutually believing that they are completing it. A joint commitment in turn is defined as a joint persistent goal (JPG). The team T's JPG to achieve p, where p stands for completion of a team action, is denoted $(JPG\ T\ p\ q)$. The variable q is a relevance term and is true if and only if p is still relevant; if the team mutually believes q to be false, then there is no need to achieve p (i.e., no need to perform the team action) and so the JPG can be abandoned. For illustrative purposes, only teams with two members x and y will be considered here, with their JPG to achieve p with respect to q denoted $(JPG\ x\ y\ p\ q)$. The following definitions can be extended in a straightforward manner to larger teams.

The joint intentions framework uses temporal operators such as $\Diamond$ (eventually) and $\Box$ (always), individual propositional attitude operators such as $(BEL\ x\ p)$ and $(GOAL\ x\ p)$ (agent x has p as a belief and as a goal, respectively), and joint propositional attitude operators such as $(MB\ x\ y\ p)$ and $(MG\ x\ y\ p)$ (agents x and y have p as a mutual belief and as a mutual goal, respectively) to build more complex modal operators to describe both individual and team mental states. Two other operators, the weak achievement goal (WAG) operator and the weak mutual goal (WMG) operator, are needed to define a JPG.

WeakAchievementGoal

$$
\begin{aligned}
(WAG\ x\ y\ p\ q) \quad \triangleq \quad & (\neg(BEL\ x\ p) \wedge (GOAL\ x\ \Diamond p)) \vee \\
& [(BEL\ x\ p) \wedge (GOAL\ x\ \Diamond(MB\ x\ y\ p))] \vee \\
& [(BEL\ x\ \Box\neg p) \wedge (GOAL\ x\ \Diamond(MB\ x\ y\ \Box\neg p))] \vee \\
& [(BEL\ x\ \neg q) \wedge (GOAL\ x\ \Diamond(MB\ x\ y\ \neg q))]
\end{aligned}
$$

An agent x on a team with another agent y will have p as a WAG with respect to q when at least one of four conditions holds:

1: x does not believe that p has been achieved, and x has as a goal for p to be achieved;

2: x believes that p has been achieved, and has as a goal for the team to mutually believe that p has been achieved;

3: x believes that p is unachievable, and has as a goal for the team to mutually believe that p is unachievable; or

4: x believes that p is irrelevant, and has as a goal for the team to mutually believe that p is irrelevant.

Notice that the first condition merely requires that x not believe that p has been achieved; it is not necessary for x to believe that p has not been achieved.

WeakMutualGoal

$$
(WMG\ x\ y\ p\ q) \quad \triangleq \quad (MB\ x\ y\ (WAG\ x\ y\ p\ q) \wedge (WAG\ y\ x\ p\ q))
$$

A team with members x and y has p as a WMG with respect to q when there is a mutual belief among team members that each team member has p as a WAG.

JointPersistentGoal

$$(JPG\ x\ y\ p\ q) \triangleq (MB\ x\ y\ \neg p) \wedge (MG\ x\ y\ p) \wedge (UNTIL\ [(MB\ x\ y\ p) \vee (MB\ x\ y\ \Box\neg p) \vee (MB\ x\ y\ \neg q)]\ (WMG\ x\ y\ p\ q))$$

In order for a team with members x and y to have p as a JPG with respect to q, four conditions must hold:

1: All team members mutually believe that p is currently unachieved;

2: All team members have p as their mutual goal, i.e., they mutually know that they want p to be true eventually; and

3: Until p is mutually known to be achieved, unachievable or irrelevant, the team holds p as a WMG.

To enter into a joint commitment (JPG) in the first place, all team members must establish appropriate mutual beliefs and commitments. The commitment to attain mutual belief in the termination of p is a key aspect of a JPG. This commitment ensures that team members stay updated about the status of team activities, and thus do not unnecessarily face risks or waste their time.

These principles are embodied in Machinetta in the following way. When a team plan is instantiated, the proxies may communicate with their respective RAPs about whether to participate in the plan. Upon successfully triggering a new plan, the proxies perform the "establishJointCommitment" procedure specified by their coordination policy to ensure that all proxies agree on the plan. Because each proxy maintains separate beliefs about these joint goals, the team can detect (in a distributed manner) any inconsistencies among team

members' plan beliefs. The proxies then use termination conditions, specified in the TOP, as the basis for automatically generating the communication necessary to jointly terminate a team plan when those conditions are met.

3.3 Role Allocation

Roles are slots for specialized execution that the team may potentially fill at runtime. Assignment of roles to team members is of critical importance to team success. This is especially true for heterogeneous teams, where some team members have little or no capability to perform certain roles. However, even for homogeneous teams, team members can usually only perform a limited number of roles simultaneously and so distributing roles satisfactorily throughout the team is of great importance.

Upon instantiation of a newly triggered plan, Machinetta proxies also instantiate any associated roles. The initial plan specification may name particular team members to fill these roles, but often the roles are unfilled and are then subject to role allocation. The proxies themselves have no ability to achieve goals at the domain level; instead, they must ensure that all of the requisite domain-level capabilities are brought to bear by informing team members of their responsibility to perform instantiated roles that are allocated to them. One role allocation algorithm successfully used in Machinetta is described in Section 5.

3.4 Example

To see how joint intentions and role allocation affect team behavior, consider an example of personal assistant proxies in an office environment. A group of three researchers, Scientist1, Scientist2, and Scientist3, need to make a joint presentation of their work at a meeting. Each person has a proxy (Proxy1 for

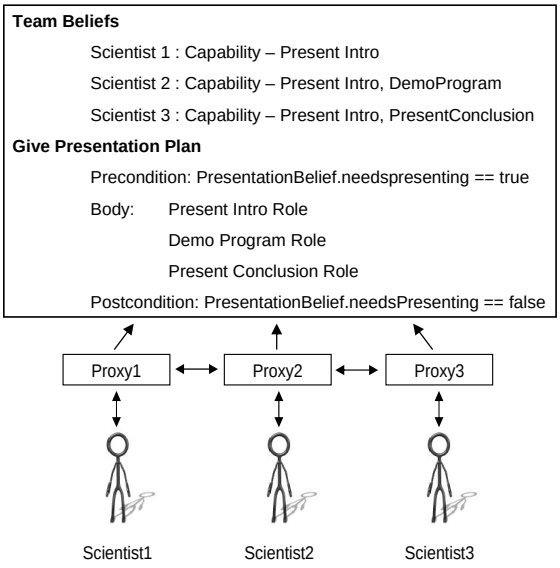


Figure 2: Office assistant TOP and architecture.

Scientist1, etc.) that facilitates his participation in team plans. The task of making the presentation together is represented by a team plan, which is shared by all the proxies in a TOP as seen in Figure 2. The presentation involves multiple roles which should be allocated to different group members.

The team plan is instantiated once the belief exists that there is a presentation that needs to be done. Only one proxy considers taking on a role at a time in order to eliminate redundancy of plan roles. At the time of consideration, the proxy can either ask the person it represents if that role should be taken or the proxy can decide autonomously whether or not the role should be accepted. If the proxy decides to act autonomously, it determines whether to accept the role by estimating a capability level of the person, based on the person's ability to do the task and how many roles that person currently has. If that capability level is higher than a threshold that is set for that particular role, the proxy accepts the role and notifies the person. Otherwise, the role is rejected and passed on

to another proxy in the hopes of it being allocated to someone more capable.

For the purposes of this example, suppose that the roles are successfully allocated, with Scientist1 presenting the introduction, Scientist2 presenting the demonstration, and Scientist3 presenting the conclusion. The researchers begin preparing their respective portions of the presentation. The proxies all have the JPG of making the presentation.

Now consider four ways in which this joint commitment can be terminated. In the first case, suppose that the meeting time arrives and the three scientists present their respective portions. As each completes his part of the presentation, his proxy is updated of the status. Once Proxy3 is notified that the conclusion has been presented, it knows that the presentation has been completed and so the JPG has been achieved. It now communicates this fact to the other proxies, so that all members of the team mutually believe that the presentation has been completed.

In the second case, suppose that Scientist3 becomes sick on the day of the presentation. He informs his proxy that he will be unable to attend. Proxy3 realizes that without Scientist3's participation the JPG is unachievable, and so it drops its goal of making the presentation. Under its joint commitment, it then communicates this information to the other proxies, who can then notify their users. This allows team members to stop preparations for the presentation and attend to other business. Once mutual belief that the goal is unachievable is established, the joint commitment dissolves. Because Scientist3 is the only team member capable of presenting the conclusion, there is no way to salvage the team plan.

The third case is similar to the second, but it is Scientist1 who falls ill. Proxy1 then notifies Proxy2 and Proxy3 that the goal is unachievable, and so they drop the JPG. In this case, however, Proxy2 and Proxy3 recognize that it

may be possible to still make the presentation; Proxy2 and Proxy3 then enter into a new joint commitment to repair the team plan. They do so by reallocating the introduction presentation to someone other than Proxy1; for the sake of this example, say that Proxy2 accepts this role. The new, repaired team plan can now be instantiated and Proxy2 and Proxy3 enter into a JPG to perform the presentation. Scientist2 is informed that he must present the introduction as well as the demonstration, and the meeting can go on as scheduled.

In the last case, Proxy3 learns that the meeting has been cancelled and so the presentation has become irrelevant. As a result, it drops its goal of presenting, and the JPG of presenting becomes false as well. However, as in the case of the goal being unachievable, the team behavior is not completely dissolved, because only Proxy3 knows that the presentation is irrelevant; a WAG to make the presentation persists. Proxy3 now must take action to achieve mutual belief among all team members that the presentation is irrelevant. To achieve this, it notifies the other two proxies that the meeting has been cancelled. These proxies in turn notify their users of the cancellation. Only when there is mutual belief that the presentation is irrelevant are the proxies fully released from their joint commitment.

4 Domains

The proxy approach has been applied earlier to several domains such as battlefield simulations (Tambe, 1997b) and RoboCup soccer simulations (Pynadath & Tambe, 2003; Kitano et al., 1997). This section will describe three additional domains that have been used to explore proxy-based teamwork. In each of these domains the same teamwork approach has been applied and been shown to be effective without changes to the key ideas.

The first domain is that of a team of personal assistant agents. Individual

software agents embedded within an organization represent each human user in the organization and act on their behalf. These personal assistant agents work together in teams toward service of cooperative tasks. Such agentified organizations could potentially revolutionize the way a variety of tasks are carried out by human organizations. In an earlier research project called "Electric Elves", an agent system was deployed at USC with a small number of users and ran continuously for nine months (Chalupsky, Gil, Knoblock, Lerman, Oh, Pynadath, Russ, & Tambe, 2002). The longest running multiagent system in the world, it provided personal assistant agents (proxies) for about a dozen researchers and students and integrated several schedulers and information agents. The resulting small-scale team of 15-20 agents aided in daily tasks such as rescheduling meetings, selecting presenters for research meetings, tracking people and ordering meals. Communicating with palm pilots and cell phones, the personal assistant agents adjusted their autonomy appropriately. Partly building on this experience, work has begun work on a more comprehensive joint project with SRI International that is known as CALO. The aim of CALO is to create a wide-ranging and functional personal assistant agent that maintains a persistent presence by continuously learning from its user and acting on its user's behalf. Designing a ubiquitous agent that propagates its utilization by providing incentives to both institutions and individuals critically depends on the development of efficient and effective teamwork.

In the second domain, disaster response (see Figure 3), teams are created to leverage the unique capabilities of Robots, Agents and People (RAPs). Proxy-facilitated teamwork is vital to effective creation of RAP teams. A major challenge stems from the fact that RAP entities may have differing social abilities and hence differing abilities to coordinate with their teammates. In order to fully model these challenges, the experimental platform in this project is an

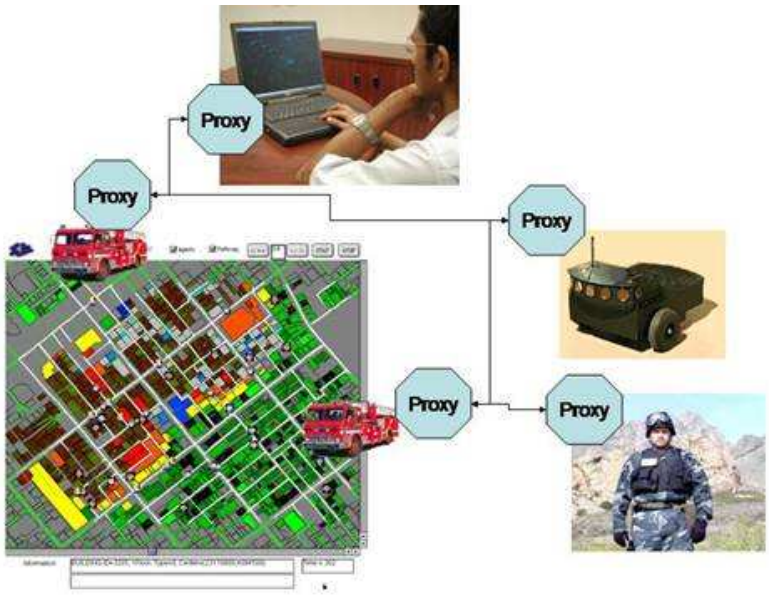


Figure 3: Disaster Response using Machinetta Proxies.

extension of the RoboCup Rescue simulation environment (Kitano, Tadokoro, Noda, Matsubara, Takahashi, Shinjoh, & Shimada, 1999) that enables human-robot interactions. Fire brigade agents act in a virtual city, while human and robot team members act in the physical world. The fire brigades can search the city after an earthquake has hit and can extinguish any fires that are found. The agents try to allocate themselves to fires in a distributed manner, but can call on the expertise of the human fire chief if required. The fire chief can allocate trucks to fires easily both because of a more global view of the situation and because the spatial, high-level reasoning required is well suited to human capabilities. Thus, the fire chief's proxy must carefully adjust its own autonomy when accepting and rejecting roles.

The third domain involves a type of Unmanned Aerial Vehicle (UAV) known as Wide Area Search Munitions (WASMs), which are part UAV and part mu-

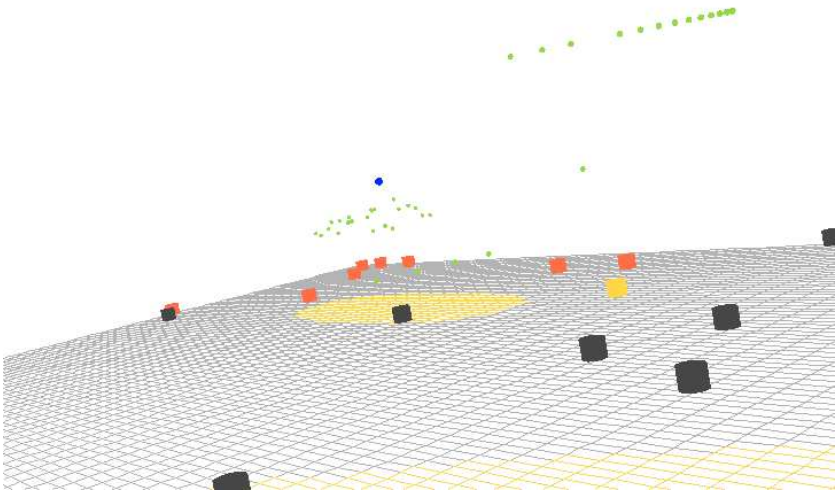


Figure 4: Unmanned Aerial Vehicle Simulator.

nition (Scerri, Xu, Liao, Lai, & Sycara, 2004). Experiments were performed using a simulation environment. Figure 4 shows a screenshot of the simulation environment in which a large group of WASMS (small spheres) are flying in protection of a single aircraft (large sphere). Various surface to air missle sites are scattered around the environment. Terrain type is indicated by the color of the ground. As many as 200 WASMs were simulated, each with its own Machinetta proxy. In the experiments, a team of WASMs coordinate to find and destroy ground based targets in support of a manned aircraft that they are guarding.

5 Novel Role Allocation Method

To allocate unfilled roles to team members, a novel role allocation algorithm has been developed that draws upon ideas from distributed constraint optimization problems (DCOPs). Based on valued constraints, DCOP is a powerful and natural representation for the role allocation problem. Mapping the problem to

a well-known paradigm like DCOP allows a large body of work to be leveraged for the algorithm. DCOP-based algorithms have been previously applied to limited role allocation problems, but have several shortcomings when used for very large teams in dynamic environments. The DCOP-based role allocation algorithm for teams, Low-communication, Approximate DCOP (LA-DCOP), is designed to overcome these shortcomings in extreme teams.

Details of the LA-DCOP algorithm are provided in the following two sections. First, a formal description role allocation problem is presented. The second section presents the LA-DCOP algorithm and describes how it solves a DCOP representation of the role allocation problem.

5.1 Problem Description

Simple role allocation problems for a single point in time can be formulated as a generalized assignment problem (GAP), which is a well known representation. Under this formulation, roles are assigned to team members, subject to resource constraints, yielding a single, static allocation. GAP is must be extended to include more complex aspects of role allocation such as dynamism. The solution of this extended GAP (E-GAP) is a series of allocations through time. LA-DCOP solves a DCOP representation of the E-GAP. The next two subsections provide formal descriptions of GAP and E-GAP.

5.1.1 GAP

A GAP problem adapted for role allocation is defined by team members for performing roles and roles to be assigned (Shmoys & Tardos, 1993). Each team member, $e_i \in E$, is defined by its capability to perform roles, $R = \{r_1, \ldots, r_n\}$, and their available resources. The capability of a team member, e_i, to perform a role, r_i, is quantitatively given by: $Cap(e_i, r_i) \rightarrow [0, 1]$. Capability reflects the

quality of the output, the speed of task performance or other factors affecting output. Each role requires some resources of the team member in order to be performed. Resource requirements of a team member e_k for a role r_j are written as $Resources(e_k, r_j)$ and the available resources of an agent, e, as $e.resources$.

Following convention, we define a matrix A, where $a_{i,j}$ is value of the ith row and jth column and

$$a_{i,j} = 1 \text{ if } e_i \text{ is performing } r_j \text{ otherwise } a_{i,j} = 0.$$

Thus, the matrix A defines the allocation of roles to team members. The goal in GAP is to maximize:

$$f(A) = \sum_{e \in E} \sum_{r \in R} Cap(e, r) \times a_{e,r}$$

such that

$$\forall i(\forall e \in E(\sum_{r \in R} Resources(e, r) \times a_{e,r} \leq e.resources))$$

and

$$\forall r \in R \sum_{e \in E} a_{e,r} \leq 1.$$

Intuitively, this says that the goal is to maximize the capabilities of the agents assigned to roles, subject to the resource constraints of team members, ensuring that at most one team member is assigned to each role but potentially more than one role per team member.

5.1.2 Extended GAP

To introduce the dynamics of extreme teams into GAP, make R, E, Cap and $Resources$ functions of time. The most important consequence of this is that a single allocation A is no longer sufficient; rather, a sequence of allocations is needed, $A^{\rightarrow}$, one for each discrete time step. A delay cost function, $DC(r_i, t)$, captures the cost of not performing r_i at time t. Thus, the objective of the E-GAP problem is to maximize:

$$f(A^{\rightarrow}) = \sum_{t} \sum_{e \in E} \sum_{r \in R} (Cap(e, r, t) \times a_{e,r,t}) - \sum_{t} \sum_{r \in R} (1 - \sum_{e \in E} a_{e,r,t}) \times DC(r, t)$$

such that

$$\forall i (\forall e \in E (\sum_{r \in R} Resources(e, r) \times a_{e,r,t} \leq e.resources))$$

and

$$\forall r \in R \sum_{e \in E} a_{e,r,t} \leq 1$$

Thus, extreme teams must allocate roles rapidly to accrue rewards, or else incur delay costs at each time step.

5.2 LA-DCOP

Given the response requirements for agents in extreme teams, they must solve E-GAP in an approximate fashion. LA-DCOP is a DCOP algorithm that is being proposed for addressing E-GAP in a distributed fashion. LA-DCOP exploits key properties of extreme teams that arise due to large-scale domains and similarity of agent functionality (e.g., using probability distributions), while simultaneously addressing special role-allocation challenges of extreme teams

(e.g., inability of strong decomposition into smaller subproblems). In DCOP, each agent is provided with one or more variables and must assign values to variables (Fitzpatrick & Meertens, 2001; Zhang & Wittenburg, 2002; Modi, Shen, & Tambe, 2002). LA-DCOP maps team members to variables and roles to values, as shown in Algorithm 1. Thus, a variable taking on a value corresponds to a team member taking on a role. Since team members can take on multiple roles simultaneously, each variable can take on multiple values at once, as in graph multi-coloring.

In E-GAP, a central constraint is that each role should be assigned to only one team member, which corresponds to each value being assigned by only one variable. In DCOP, this requires having a complete graph of *not equals* constraints between variables (or at least a dense graph, if not strictly E-GAP) – the complete graph arises because agents in extreme teams have similar functionality. Dense graphs are problematic for DCOP algorithms (Modi et al., 2002; Fitzpatrick & Meertens, 2001), so a novel technique is required. For each value, create a *token*. Only the team member currently holding a token representing a value can assign that value to its variable. If the team member does not assign the value to its variable, it passes the token to a teammate who then has the opportunity to assign the value represented by the token. Essentially, tokens deliberately reduce DCOP parallelism in a controlled manner. The advantage is that the agents do not need to communicate to resolve conflicts.

Given the token-based access to values, the decision for the agent becomes whether to assign values represented by tokens it currently has to its variable or to pass the tokens on. First the agent must check whether the value can be assigned while respecting its local resource constraints (Algorithm 1, line 10). If the value cannot be assigned within the resource constraints of the team member, it must choose a value(s) to reject and pass on to other teammates in the

form of a token(s) (Algorithm 1, line 13). The agent keeps values that maximize the use of its capabilities (performed in the MaxCap function, Algorithm 1, line 11). Notice that changing values corresponds to changing roles and may not be without cost. Also notice that the agent is "greedy" in that it performs the roles it is best at.

```
Algorithm 1: VarMonitor(Cap, Resources)
(1)   V ← ∅
(2)   while true
(3)     msg ← getMsg()
(4)     token ← msg
(5)     if token.threshold = NULL
(6)       token.threshold ← ComputeThreshold(token)
(7)     if token.threshold < Cap(token.value)
(8)       V ← V ∪ token.value

(10)      if Σ_{v∈V} Resources(v) ≥ agent.resources
(11)        out ← V− MaxCap(Values)
(12)        foreach v ∈ out
(13)          PassOn(new token(v))
(14)          Values ← Values − out

(16)    else
(17)      PassOn(token) /* Cap < threshold */
```

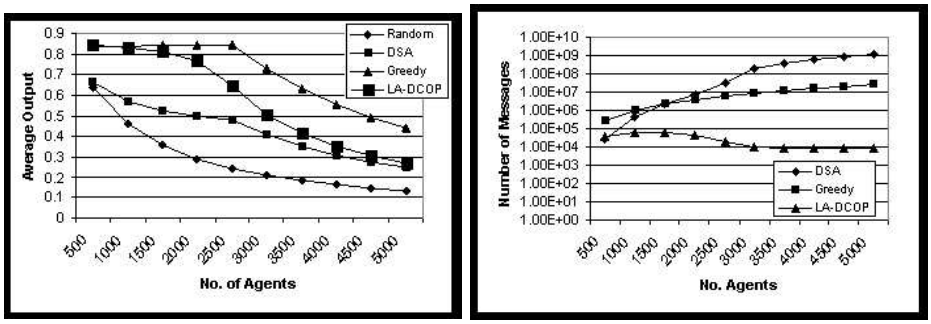
Secondly, a team member must decide whether it is in the best interests of the team for it to assign the value represented by a token to its variable (Algorithm 1, line 7). The key question is whether passing the token on will lead to a more capable team member taking on the role. Using probabilistic models of the members of the team and the roles that need to be assigned, the team member can choose the minimum capability the agent should have in order to assign the value. Notice that it is the similar functionality of the agents in extreme teams and their large numbers that allows us to apply probabilistic models. Intuitively, the agent estimates the likely capability of an agent performing this role in a good allocation. This minimum capability is referred to as the *threshold*. The threshold is calculated once (Algorithm 1, line 6), and attached to the token

as it moves around the team. Computing thresholds that maximize expected utility is a key part of this algorithm; once thresholds are calculated, agents simply circulate tokens until each token is held by an agent with capability above threshold for the role and within resource constraints. (To avoid agents passing tokens back and forth, each token maintains the list of agents it has visited; if all agents have been visited, the token can revisit agents, but only after a small delay.)

6 Experiments

LA-DCOP has been tested extensively in three environments. The first is an abstract simulator that allows many experiments to be run with very large numbers of agents (Okamoto, 2003). In the simulator, agents are randomly given capabilities for each type of role, with some percentage being given zero capability. Given many agents with overlapping capabilities for role types, dense constraint graphs result, where a constraint ensures two agents do not take the same role. For each time step that the agent has the role, the team receives ongoing reward based on the agent's capability. Message passing is simulated as taking one time step and messages always get through. New roles appear spontaneously and the corresponding tokens are distributed randomly. The new roles appear at the same rate that old roles disappear, hence keeping the total number of roles constant. Each data point represents the average from 20 runs.

The first experiments tests LA-DCOP against three competitors. The first is DSA, which is shown to outperform other approximate DCOP algorithms in a range of settings (Modi et al., 2002; Fitzpatrick & Meertens, 2001); empirically determined best parameters were used for DSA (Zhang & Wittenburg, 2002). DSA does not easily allow multiple roles to be assigned to a single agent, so the



(a) **(b)**

Figure 5: (a) comparing the average output per agent per time step versus the number of agents. (b) the number of messages sent versus the number of agents

comparison focuses on the case where each agent can take only one role. As a baseline, LA-DCOP is also compared against a centralized algorithm that uses a "greedy" assignment (Castelpietra, Iocchi, Nardi, Piaggio, Scalzo, & Sgorbissa, 2002) and against a random assignment. Figure 5(a) shows the relative performance of each algorithm. The experiment used 2000 roles over 1000 time steps. The y-axis shows the total reward per agent per time step, while the x-axis shows the number of agents. Not surprisingly, the centralized algorithm performs best and the random algorithm performs worst. Of the distributed algorithms, LA-DCOP performs statistically better than DSA. However, the real key is the amount of communication used, as shown in Figure 5(b). Notice that the y-axis is a logarithmic scale, thus LA-DCOP uses approximately three orders of magnitude fewer messages than the greedy algorithm and four orders of magnitude fewer messages than DSA. Thus, LA-DCOP performs better than DSA despite using far less communication, and only marginally worse than a centralized approach despite using only a tiny fraction of the number of messages.

Figure 6 shows how the performance of LA-DCOP scales with the number of agents in the system. The y-axis shows the output per agent per time step

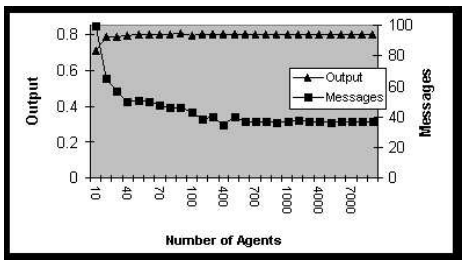


Figure 6: The average output per agent per time step (left hand y-axis) and number of messages per agent (right hand y-axis) as the number of agents is scaled up.

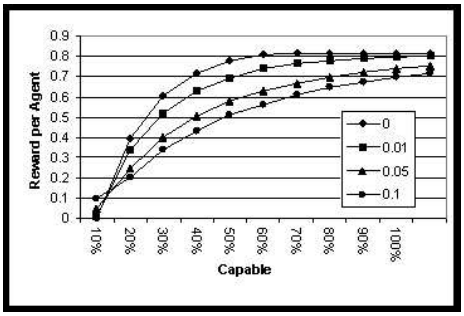
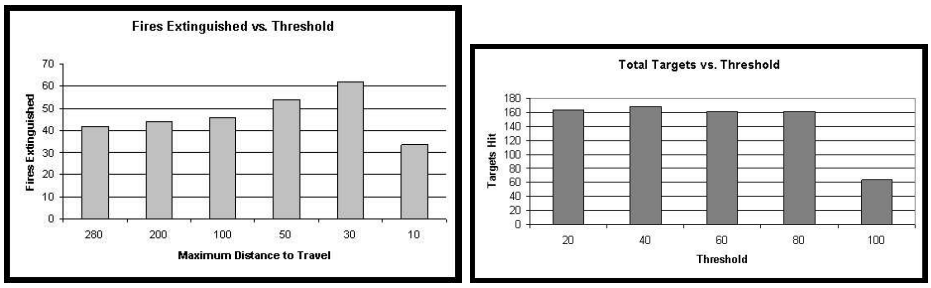


Figure 7: The effects of different proportions of roles changing each step. The y-axis shows the output per agent per time step, x-axis shows the percentage of agents with capability > 0.

(left-hand side) and average number of messages per agent (right-hand side) and the x-axis shows the number of agents. Notice that the algorithm's poorest performance is actually when the number of agents is fairly small. This is because the probability models are "less reliable" for small numbers of agents. However, for large numbers of agents, the number of messages per agent and performance per agent stays constant, suggesting that LA-DCOP can be applied to very large extreme teams. While these results are a pleasant surprise, the scope of their application – rapid role allocation for extreme teams – should be noted.

A key feature of extreme teams domains is that the roles to be assigned change rapidly and unpredictably. In Figure 7, LA-DCOP is shown to perform



(a) **(b)**

Figure 8: (a) shows the number of fires extinguished by 200 fire trucks versus threshold (b) shows the number of targets hit by UAVs versus threshold.

well even when the change is very rapid. The four lines represent different rates of change, with 0.01 meaning that every time step (i.e., the time it takes to send one message), 1% of all roles are replaced with roles requiring a different capability. At middling capability (50%), with 1% dynamics, LA-DCOP loses 10% of reward per agent on average, but complete DCOP algorithms today cannot even handle dynamics.

The second set of experiments used 200 LA-DCOP enhanced versions of Machinetta proxies (Scerri et al., 2003), distributed over a network, executing plans in two simple simulation environments. This was possibly larger than any previously published report on complex multiagent teams, and certainly an order of magnitude jump over the last published reports of teamwork based on proxies (Scerri et al., 2003). Previous published techniques for role allocation in the proxies fail to scale up to extreme teams of 200 agents — complete DCOP fails on dense graphs, and symbolic matching ignores quantitative information. The proxies execute sophisticated teamwork algorithms as well as LA-DCOP and thus provide a realistic test of LA-DCOP. The first environment is a version of a disaster response domain where fire trucks must fight fires. Capability in this case is the distance of the truck from the fire, since this affects the time until the fire is extinguished. Hence, in this case, the threshold corresponds

to the maximum distance the truck will travel to a fire. Figure 6(a) shows the number of fires extinguished by the team versus threshold. Increasing thresholds initially improves the number of fires exstinguished, but too high a threshold results in a lack of trucks accepting roles and a decrease in performance. In the second domain, 200 simulated UAVs explore a battle space, destroying targets of interest. While in this domain LA-DCOP effectively allocates roles across a large team, thresholds are of no benefit. The key point of these experiments is to show that LA-DCOP can work effectively in a fully distributed environment with realistic domains and large teams.

7 Summary

This chapter reports on Machinetta, a proxy-based approach to enabling teamwork among diverse entities. This approach is implemented in Java and is derived from an earlier model, STEAM, that was implemented in Soar. The Machinetta proxies are equip each team member with a model of the commitments and responsibilities necessary for teamwork. This model is derived from a BDI framework and the notion of joint intentions. These proxies have been effectively applied to a variety of domains ranging from personal assistants to disaster rescue to Unmanned Aerial Vehicles (UAVs). Across each of these domains, a key challenge that these proxies must attack is role allocation. These Machinetta proxies and the BDI framework have led to the creation of a new role-allocation algorithm (LA-DCOP). This innovation has allowed for the construction of proxies that have repeatedly and definitively demonstrated effective teamwork in diverse domains.

Appendix A

Soar Communication Rules:

Step 1: The rules in file create-communicative-goals are used to match an agent's private state (beliefs) with any of the team operator's termination conditions – i.e., conditions that would make the team operator achieved, unachievable or irrelevant. These communicative goals are only possible as communicative goals at this juncture.

Step 2: The rules in file terminate-jpg-estimate-tau are used to estimate the likelihood that the given communicative goals are the common knowledge in the team. The likelihood is specified as high, low or medium.

Step 3: The rules in file elaborate-communicative-goals are used to match the specified likelihoods with the communication costs to check if communication is possible

Step 4: If communication is possible, rules in communicate-private-beliefs are used to communicate the relevant information to others in the team.

Step 5: Due to communication or high likelihood that relevant information is mutually believed, agents assume that certain knowledge is now mutually believed.

References

Castelpietra, C., Iocchi, L., Nardi, D., Piaggio, M., Scalzo, A., & Sgorbissa, A. (2002). Coordination among heterogenous robotic soccer players. *Proceedings of International Conference on Intelligent Robots and Systems 2002.*

Chalupsky, H., Gil, Y., Knoblock, C. A., Lerman, K., Oh, J., Pynadath, D. V., Russ, T. A., & Tambe, M. (2002). Electric Elves: Agent technology for supporting human organizations. *AI Magazine*, *23*(2), 11–24.

Cohen, P. R., & Levesque, H. J. (1991). Teamwork. *Nous*, *25*(4), 487–512.

Fitzpatrick, S., & Meertens, L. (2001). *Stochastic algorithms: Foundations and applications, proceedings saga 2001*, Vol. LNCS 2264, Chap. An Experimental Assessment of a Stochastic, Anytime, Decentralized, Soft Colourer for Sparse Graphs, pp. 49–64. Springer-Verlag.

Georgeff, M., Pell, B., Pollack, M., Tambe, M., & Wooldrige, M. (1998). The belief-desire-intention model of agency. *Proceedings of Agents, Theories, Architectures and Languages (ATAL)*.

Grosz, B., & Kraus, S. (1996"). Collaborative plans for complex group actions. *Artificial Intelligence*, *86*, 269–358.

Hill, R., Chen, J., Gratch, J., Rosenbloom, P., & Tambe, M. (1997). Intelligent agents for the synthetic battlefield: A company of rotary wing aircraft. *Innovative Applications of Artificial Inttigence (IAAI-97)*.

Jennings, N. (1995). The archon systems and its applications. Project Report.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., & Matsubara, H. (1997). RoboCup: A challenge problem for AI. *AI Magazine*, *18*(1), 73–85.

Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjoh, A., & Shimada, S. (1999, October). Robocup rescue: Searh and rescue in large-scale disasters as a domain for autonomous agents research. *Proc. 1999 IEEE Intl. Conf. on Systems, Man and Cybernetics*, Vol. VI (pp. 739–743). Tokyo.

Modi, P. J., Shen, W., & Tambe, M. (2002). *Distributed constraint optimization and its application* (Technical Report ISI-TR-509). University of Southern California/Information Sciences Institute.

Nair, R., Ito, T., Tambe, M., & Marsella, S. (2002). Task allocation in robocup rescue simulation domain. *Proceedings of the International Symposium on RoboCup*.

Newell, A. (1990). *Unified theories of cognition.* Cambridge, Massachusetts: Harvard University Press.

Okamoto, S. (2003). Dcop in la: Relaxed. Master's thesis, University of Southern California.

Pynadath, D. V., & Tambe, M. (2003). An automated teamwork infrastructure for heterogeneous software agents and humans. *Journal of Autonomous Agents and Multi-Agent Systems, Special Issue on Infrastructure and Requirements for Building Research Grade Multi-Agent Systems*, 7:71–100.

Scerri, P., Pynadath, D. V., Johnson, L., P., R., Schurr, N., Si, M., & Tambe, M. (2003). A prototype infrastructure for distributed robot-agent-person teams. *The Second International Joint Conference on Autonomous Agents and Multiagent Systems.*

Scerri, P., Xu, Y., Liao, E., Lai, G., & Sycara, K. (2004). Scaling teamwork to very large teams. *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems.* submitted.

Shmoys, D., & Tardos, E. (1993). An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, *62*, 461–474.

Tambe, M. (1997a). Agent architectures for flexible, practical teamwork. *National Conference on AI (AAAI97)*, 22–28.

Tambe, M. (1997b). Towards flexible teamwork. *Journal of Artificial Intelligence Research*, *7*, 83–124.

Tidhar, G., Rao, A., & Sonenberg, E. (1996). Guided team selection. *Proceedings of the Second International Conference on Multi-Agent Systems.*

Werger, B. B., & Mataric, M. J. (2000). Broadcast of local eligibility for multi-target observation. *Proc. of 5th Int. Symposium on Distributed Autonomous Robotic Systems (DARS).*

Yen, J., Yin, J., Ioerger, T. R., Miller, M. S., Xu, D., & Volz, R. A. (2001). Cast: Collaborative agents for simulating teamwork. *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 1135–1142).

Zhang, W., & Wittenburg, L. (2002). Distributed breakout revisited. *Proceedings of American Association for Artificial Intellgince 2002.*