A Prototype Infrastructure for Distributed Robot-Agent-Person Teams

Paul Scerri, David Pynadath, Lewis Johnson, Paul Rosenbloom, Mei Si,
Nathan Schurr and Milind Tambe
Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
{scerri,pynadath,johnson,rosenbloom,meisi,schurr,tambe}@isi.edu

ABSTRACT

Effective coordination of robots, agents and people promises to improve the safety, robustness and quality with which shared goals are achieved by harnessing the highly heterogeneous entities' diverse capabilities. Proxy-based integration architectures are emerging as a standard method for coordinating teams of heterogeneous entities. Such architectures are designed to meet imposing challenges such as ensuring that the diverse capabilities of the group members are effectively utilized, avoiding miscoordination in a noisy, uncertain environment and reacting flexibly to changes in the environment. However, we contend that previous architectures have gone too far in taking coordination responsibility away from entities and giving it to proxies. Our goal is to create a proxy-based integration infrastructure where there is a beneficial symbiotic relationship between the proxies and the team members. By leveraging the coordination abilities of both proxies and socially capable team members the quality of the coordination can be improved. We present two key new ideas to achieve this goal. First, coordination tasks are represented as explicit *roles*, hence the responsibilities not the actions are specified, thus allowing the team to leverage the coordination skills of the most capable team members. Second, building on the first idea, we have developed a novel role allocation and reallocation algorithm. These ideas have been realized in a prototype software proxy architecture and used to create heterogeneous teams for an urban disaster recovery domain. Using the rescue domain as a testbed, we have experimented with the role allocation algorithm and observed results to support the hypothesis that leveraging the coordination capabilities of people can help the performance of the team.

1. INTRODUCTION

Robots, agents and people have a diverse set of capabilities and characteristics. Effectively harnessing these capabilities in joint activity promises to improve the safety, quality and robustness with which some goals can be achieved. For example, an agent's ability to vigilently process large amounts of information, a person's ability to solve problems, and an "expendable" robot's ability to go where people cannot safely go, could be combined to make urban disaster response safer and more effective. However, enabling multiple hetrogenous entities to coordinate on real-world tasks is difficult, especially due to challenges such as ensuring robust execution in the face of a dynamic environment, providing abstract task specifications without the low-level coordination details, and assigning appropriate agents to specific tasks[16, 1].

The approach taken in this work is to provide each RAP[1] with a *proxy* and have the proxies act together in a *team*. There is an emerging consensus in the field that teamwork can enable flexible, robust coordination among multiple heterogeneous entities and allow them to achieve their shared goals. Furthermore, previous work has also illustrated the power of domain-independent team coordination algorithms to realize such flexible teamwork and of the idea of proxies to enable diverse agents to work together in a team [7, 16, 18, 5]. Existing proxies were initially designed to enable a group of "non-team-ready" software agents to work together in a team and were later extended to allow humans to work together in the same team. For example, in the Electric Elves project, proxies helped a team of researchers with routine coordination activities[2].

However, previous approaches have two key limitations when applied to RAP teams. First, the proxies performed all *coordination* tasks without the possibility of getting input from team members. This precluded more socially capable RAPs from utilizing their *coordination* skills for the good of the team. For example, people who may be able to quickly resolve resource conflicts could not help proxies that found the problem difficult. A better approach is an infrastructure that takes over routine coordination tasks and aids RAPs without social capabilities, while still allowing socially capable RAPs to freely provide their coordination preferences and skills. The result will be an infrastructure that synergistically combines the coordination capabilities of RAPs and proxies to the benefit of the team.

In particular, in previous work, the RAPs were essentially considered to have expertise only in domain-level tasks. Thus, these previous approaches treated domain tasks and coordination tasks differently. The domain tasks, i.e., those for actually achieving the team's goals, are represented as *roles* and allocated to the RAPs most capable of completing them. For these domain roles, the RAP has wide flexibility in what decisions and actions are taken to fulfill the role and can act in a way that best utilizes its particular capabilities. Conversely, coordination tasks are performed exclusively by proxies following some predefined rules, without any possibility of input from RAPs.

A key idea in this work is to transform coordination tasks into explicit roles and to allow either proxies or RAPs to perform the roles. A potential pitfall with making all coordination tasks into roles is

[1] We use the term RAP to refer to a Robot, Agent and/or Person.

that RAPs may get overloaded performing coordination tasks. We avoid this pitfall by integrating *adjustable autonomy* reasoning into the core of the proxy. The adjustable autonomy reasoning decides whether the RAP or the proxy will perform a particular coordination role. Since proxies have the ability to perform all coordination roles, responsibility for performing the role need only be transferred to the RAP when the RAP's special abilities can perform the role better and the team will benefit from the improved role performance. Tight integration of adjustable autonomy reasoning into a proxy and its use for determining responsibility for coordination tasks differentiates this work from previous adjustable autonomy work[10].

A second limitation of previous approaches is that the role allocation algorithms are typically designed for small teams with reliable, consistent communication [22, 20]. The algorithms are often not suited to highly dynamic, large scale RAP teams. Many of the role allocation algorithms are designed to find optimal or near-optimal role allocations and incur unreasonable computation and communication overheads in RAP teams. For example, allocations of RAPs to roles found using contract networks [**?**], auctions [**?**] or constraints [12] will be very good, but at the expense of high communication overheads. Often, given large teams in dynamic and uncertain environments, the team will be better served by quickly finding a reasonable allocation of RAPs to roles, while severely limiting the time taken. Previous role allocation algorithms have also typically relied on knowing available RAPs and "neighbors" in advance and having an ability to communicate with specific neighbors, e.g., algorithms based on constraint satisfaction have this property [12]. We cannot rely upon such knowledge and consistent communication networks in highly dynamic teams.

To overcome the limitations of previous role allocation algorithms, we have developed a new algorithm that is distributed, has low overheads and allows role allocation and execution to occur in parallel. While the algorithm will not always result in near optimal role allocations, it is a useful tool for the proxies in some domains. The algorithm works by creating a *role-allocation role* for the coordination task of assigning the role. When such a role is assigned to a particular proxy and RAP, there must be one of two outcomes: the role is accepted and eventually performed by the RAP or responsibility for the role-allocation role is transferred to another proxy and RAP. Since the proxy only considers transferring the role to those RAPs that are potentially capable of performing a role, incapable RAPs may never know anything about it. Because how the role-allocation role is achieved is not specified, more socially capable RAPs can use exploit their sophisticated reasoning for deciding whether to accept the role, while less capable RAPs can rely on the relatively simple reasoning of their proxy to make a decision. For example, if responsibility for allocating a role of going into a burning building is given to the proxy of a person, the proxy can ask the person whether or not they wish to accept the role of going into the building.

The proxies are implemented as lightweight Java processes and can run on various devices, including handheld computers. The proxies work with the whole spectrum of RAPs, from relatively simple robots to expert human users. The design of the proxies makes them easily configurable and reusable across domains, thus providing the multiagent community with a useful piece of software for future research. RAP teams, utilizing the proxy infrastructure, are being evaluated in an urban disaster rescue domain. Current teams have up to 12 RAPs, coordinating on up to 500 roles over a period of an hour in a simulation of an earthquake-struck city. Agents control rescue vehicles, people play supervisory roles, and people and robots search buildings for survivors. The team is highly configurable, hence providing an interesting testbed for RAP research. We have conducted experiments varying two dimensions (domain-level complexity and allocation of roles to people) and examined the effects of those variations on domain-level and role-allocation performance. Our results, while by no means conclusive, indicate that allowing the proxies to leverage the coordination abilities of people improves performance.

2. DOMAIN: DISASTER RESCUE

The concept of RAP teams is a relatively new one. Hence models and theoretical frameworks for evaluating approaches are not available. The relevant algorithms need to be implemented and tested in environments as close to real environments as possible. Our current domain of interest is an urban disaster rescue, in particular providing emergency response to an earthquake. The aim is to have large teams of humans, agents and robots performing the wide variety of roles required for the response effort.

Our experimental platform is a combination of the RoboCup Rescue simulation environment[9] and significant extensions that enable human and robot interactions, (see Figure 1). Part of the scenario exists in simulation, while the rest occurs in real buildings with real robots. Simulated buildings burn, with fires spreading to adjacent buildings if they are not quickly contained. Civilians can be trapped in buildings, making the priority of fighting fire in those buildings much higher. The team's goal is to save as many trapped and injured people as possible and stop fires from spreading. 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 soon after an earthquake has hit and can extinguish any fires that are found. A human fire chief is given a high-level view of the fire-fighting progress and can make role-allocation decisions to better save civilians and limit damage. The robots can assist by maneuvering through real buildings, with simulated earthquake damage, looking for injured or trapped people. A human paramedic will work with the robot to save the trapped civilians.

Figure 1 shows the RAP team that we are developing for the earthquake domain. Fire brigades, controlled by agents, fight fires that have broken out in the city. The agents try to allocate themselves to fires in a distributed manner (see Section 4), but can call on the expertise of the human "fire chief" if required. The fire chief can allocate trucks to fires easily both because he has a more global view of the situation and because the spatial, high-level reasoning required is well suited to human capabilities. However, requiring that the Fire Chief make too many role allocation decisions can overload him and degrade performance. When civilians are trapped inside of damaged buildings, both robots and human paramedics have capabilities that can assist in saving the people. Robots are able to go into places unsafe for paramedics, while paramedics have the medical skills to help the injured people that the robots have discovered. Working together they can perform the rescue job better than either can alone.

3. RAP TEAM ARCHITECTURE

Robots, agents and people distinctly have different strengths and weaknesses. Creating teams of RAPs provides the possibility of leveraging the diverse strengths of each RAP to overcome the limitations of the others. Engaging in teamwork imposes certain constraints on the team members (e.g., informing other team members when there are successes or failures), which lead to desirable properties of group behavior[22]. Since team members have a responsibility towards each other, a team can achieve its goals robustly, with team members covering for failed teammates, supplying key

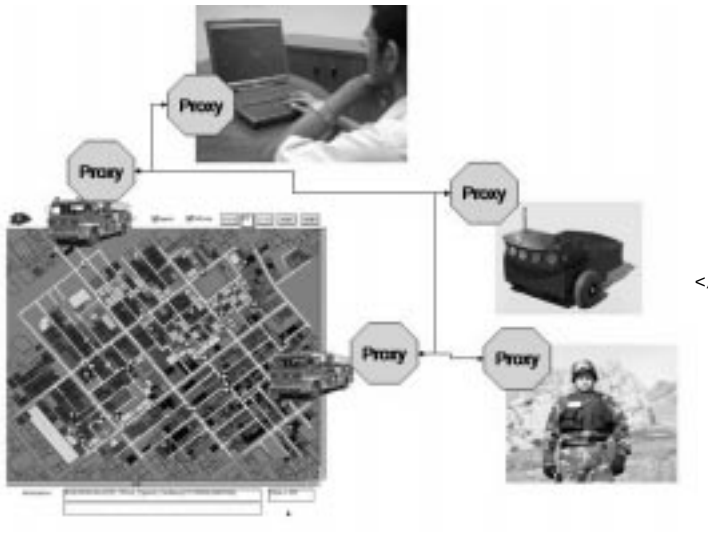

Figure 1: A proxy-based RAP team with a human "fire chief", agents controlling fire brigades (left), and paramedics and robots (right) rescuing civilians.

```
<TeamPlanTemplate Name="Fight Fire">
   <Team><Id Name="Response Team"></Id></Team>

   <Precondition>
       <Key Name="location" Type="Input"></Key>
       <Key Name="extinguished" Value="False"></Key>
   </Precondition>

   <Postcondition Type="achieved">
       <Key Name="extinguished" Type="True"></Key>
   </Postcondition>

   <Role Name="FightFire" Label="Primary"></Role>
   <Role Name="FightFire" Label="Secondary"></Role>
</TeamPlanTemplate>
```

Figure 2: Partial TOP for fire-fighting team plan.

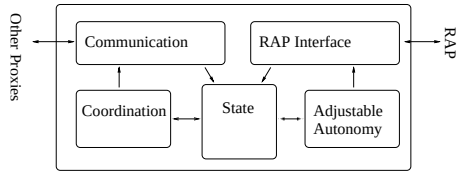

Figure 3: Proxy software architecture.

information to help each other, etc. Building on previous work, we have developed an infrastructure for coordinating RAPs that couples RAPs with proxies. In addition to performing the tasks described in previous work, such as initiating and terminating team plans, aiding recovery from failures and communicating information, our proxies have additional features that make them unique. These new features include the incorporation of adjustable autonomy reasoning (see Section 3.2), a novel role allocation algorithm (see Section 4) and the representation of coordination tasks as roles.

Teams of RAP-proxy pairs execute *Team-Oriented Programs* (TOPs) [25, 17], abstract team plans that provide high-level descriptions of the activities to be performed. TOPs specify the team's joint plans, and the inter-dependencies between those plans, but do not contain all of the details of coordination and communication required. Such programs identify the domain capabilities required to perform each role, which helps in determining how best to allocate tasks to team members. The team plans are reactively instantiated in response to events in the environment. The role allocation algorithms attempt to assign RAPs to the roles that are required in the plan. The proxies can make small changes to the plan and dynamically change assignments of tasks to RAPs to better achieve the goals. Since roles are only specified abstractly, individual RAPs are free to carry out their assigned tasks as they see fit, based upon their own capabilities. In the current implementation, TOPs are specified in XML and loaded by the proxies at runtime. Figure 2 shows a TOP, specifying a partial team plan to fight a fire in a particular building. The plan has a precondition that there is a fire at some location and a post-condition that the fire has been extinguished. There are two *Fight Fire* roles, labeled "Primary" and "Secondary", that will be filled by RAPs capable of fire fighting.

3.1 Proxies

The proxies are lightweight, domain-independent pieces of software, capable of performing the activities required to work cooperatively on TOPs[2]. The proxies are implemented in Java and are designed to run on a number of platforms including laptops and handheld devices. A proxy's software is made up of five components (see Figure 3):

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 RAP
RAP Interface: communication with the RAP

Each component abstracts away details allowing other components to make its task easier. For example, the RAP interface component is aware of what type of RAP it is connected to and the methods of interacting with the RAP, while the adjustable autonomy component deals with the RAP as an abstract entity having particular capabilities. Likewise, the communication component will be tailored to the RAP communication abilities, e.g., wireless or wired, but the coordination component will only be told available bandwidth and cost of communication. The coordination, state and adjustable autonomy modules are domain independant, while the communication and RAP Interface modules are domain specific.

A critical component in deploying the proxies is the mechanism by which they interact with their RAPs. The adjustable autonomy component is responsible for deciding what interaction should happen with the RAP, but the RAP interface component manages that interaction. The RAP interface component is the only part of the proxy that needs to be designed for a specific type of RAP. These components are very diverse, matching the diversity of the RAPs. 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 the fire brigades a simple socket communicating a small, fixed set of messages is sufficient. Since the proxies interact closely with their RAPs, it is desirable to have them in close physical proximity. For mobile RAPs, the proxies can be run on handheld devices that communicate wirelessly with robots or, in the case of a person in the field, via a graphical interface on the handheld device.

[2]The proxies are public domain software and can be downloaded from http://www.isi.edu/teamcore/doc/Machinetta

3.2 Proxy Algorithms

The beliefs in the state constitute the proxy's knowledge of the status of the team and the environment. The state is a blackboard, with components writing information to the blackboard and others reacting to information posted on the blackboard. The proxy's overall execution is message driven. When a message comes in from its RAP or from another proxy, a new belief is added to the proxy's state. Any change to the state triggers two reasoning algorithms: *Coordination* and *Adjustable Autonomy*. Either of these algorithms may in turn change the belief state, which will once again trigger the algorithms.

Algorithm 1 shows the Coordination algorithm, which instantiates the theory of *joint commitments*[3] as operationalized by STEAM. The functions, *establishJointCommitment* and *endJointCommitment* establish or terminate commitments by communicating with other proxies when a new belief triggers the start or end of a team plan. The function *communicate?* returns true if the new information about the RAPs changing capability (*CapabilityInformation*) or its progress towards achieving its role(s) (*RoleProgress*) should be communicated to others. This function encapsulates previous work on determining policies for communicating such information with team members[15].

Algorithm 1: Coordination

COORDINATION($\mathbf{B_{in}}$)
(1) **foreach** $b \in \mathbf{B_{in}}$
(2) **if** b is *CapabilityInformation* or *RoleProgress*
(3) **if** $communicate?(b)$
(4) $sendToOthers(b)$
(5) **else if** $startTeamPlan\ \alpha?(b)$
(6) $establishJointCommitment(\alpha)$
(7) ALLOCATEROLE*role*(α)
(8) **else if** $endTeamPlan\ \alpha?(b)$
(9) $endJointCommitment(\alpha)$
(10) **return** $\mathbf{B_{out}}$

The Adjustable Autonomy algorithm (Algorithm 2) is responsible for managing the interactions between the proxy and the RAP. The *if* statement beginning on Line 2 shows the basic processing that the proxy performs when its RAP is offered a role (*role offer*) or is now responsible for a new role (*new role*), e.g., by the proxy having autonomously accepted it. The $shouldRAPbeAsked?$ function is the "core" of adjustable autonomy reasoning and is responsible for deciding whether or not this particular coordination decision should be handled autonomously by the proxy or by the RAP. In the case of a role offer, it decides whether to act autonomously and, if so, decides whether or not to accept the role on behalf of the RAP (see next section for more detail).

Algorithm 2: Adjustable Autonomy

ADJUSTABLEAUTONOMY($\mathbf{B_{in}}$)
(1) **foreach** $b \in \mathbf{B_{in}}$
(2) **if** b is *role offer*
(3) **if** RAP is capable of role
(4) **if** $shouldRAPbeAsked?$
(5) Ask RAP
(6) **else if** accept autonomously?
(7) $B_{out} \leftarrow$ *role accepted*
(8) **else**
(9) $B_{out} \leftarrow$ *role rejected*
(10) **else if** b is *new role*
(11) Tell RAP it has b
(12) **return** $\mathbf{B_{out}}$

4. ROLE ALLOCATION

In this section, we focus on a central problem that motivates RAP teamwork: applying diverse RAP capabilities to tasks that suit those capabilities. To bring RAPs' special capabilities to bear, it is critical to make appropriate assignments of RAPs to roles within a team. Previous role allocation algorithms have limitations that make them inappropriate for RAP teams, especially as the teams become larger and more roles are dynamically initiated and terminated. In particular, we focus on three key weaknesses. First, previous work has not allowed RAPs with special abilities at role allocation (or any other coordination tasks) to exercise those abilities. Second, previous algorithms typically incur unacceptable computational overhead in striving for optimal allocations. For example, role-allocation algorithms based on capability analysis [24], combinatorial auctions [6], and BDI theory [19] all have computational requirements that can overly burden the team. Moreover, these algorithms often maintain a strict separation between role-allocation and role-execution phases, which is infeasible with the asynchrony inherent in large, distributed teams. Finally, some role-allocation algorithms rely on a reliable communication network and/or a consistent set of "neighbors". For example, an approach based on constraint satisfaction [12] organizes the inter-RAP communication into a tree structure that is subject to failure if individual links break down. Such algorithms typically include all the team members in the negotiation, although, in many RAP teams, only a small percentage will be able to fulfill a particular role.

We present a novel role-allocation algorithm that addresses these limitations and is thus suitable for coordinating RAP teams in highly dynamic domains. The algorithm relies on individual RAPs either accepting offered roles or passing them on to another RAP that may be able to perform the role. Notice that all roles are the shared responsibility of the team, a concept we inherit from the theory of joint intentions. The passing around of roles is done while RAPs continue with their current roles thus avoiding having a separate allocation phase. A key to this algorithm, for the purposes of RAP teams is the flexibility that a proxy has to handle a role offer. The adjustable autonomy allows the proxy to take as much or as little autonomy over coordination decisions as the situation warrants. Thus, more socially capable RAPs can utilize their abilities to make good choices about whether to accept a role and make good choices about whom to pass the role onto. There is no preset order in which roles must be passed on; thus, there is no reliance on particular communications links or RAP availability. In fact, a proxy's and RAP's knowledge of the other RAPs can be very limited, as they need only know the identity of some other RAP(s) that may potentially be able to perform the role. If a RAP is currently executing a role when offered a new role and it cannot perform both in parallel, it can take the new role if it has higher preference, capability, priority, or whatever other factors it deems important for choosing between the roles. The simple process of accepting or rejecting a role and passing it on places only minimal computation and communication requirements on the proxies and RAPs, making the algorithm appropriate for large dynamic teams. Moreover, roles need not be offered to RAPs clearly incapable of performing the roles, potentially a considerable saving. Notice, however, that this algorithm will generally not find optimal solutions.

4.1 Definition of Role Allocation Problem

In trying to allocate roles for a team of RAPs, we model each RAP as being in a space of possible states, S_i. The states of interest in this case are those dynamic features of the RAP that influence its capability to perform tasks. For example, fire brigade agent i's local state space, S_i, could consist of all possible combinations of

brigade positions and water levels. We define the set of domain-level and coordination-level roles, R, to be the set of all possible task instances (e.g., extinguishing a fire, allocating a role) that can arise during the RAP team's execution. Each such role, $r \in R$, has its own set of possible states, S_r. For example, a role associated with extinguishing the fire in a particular building would have state information representing the severity of the fire.

4.1.1 RAP Capabilities

To match RAPs to roles, we require a representation of each RAP's ability to successfully execute each role. As a starting point, we assume that the architecture has a quantitative representation of each RAP's dynamic capabilities. The heterogeneity of RAP teams requires such a quantitative representation, because although there may be many RAPs capable of filling a role, not all of these RAPs will be *equally* capable. Therefore, we need a capability specification that distinguishes among the different levels of capability among the RAPs, in addition to the different capabilities themselves.

More precisely, we represent the capabilities of a RAP i as a function, $c_i : S_i \times R \rightarrow [0, 1]$, that maps the RAP's current state and a possible role into a quantitative estimate of that RAP's ability to succeed at the given role. For example, for a fire brigade i, the capability function would represent the fact that increased distance from a building diminishes ability to put out a fire at that building in a timely fashion (e.g., $c_i(\text{pos}(i), \text{extinguish}(j)) = e^{-||\text{pos}(i) - \text{pos}(j)||}$). For the architecture to exploit such capability information in allocating roles, the proxies must have up-to-date knowledge of each RAP's current local state.

4.1.2 Role Priorities

Just as the heterogeneity of RAP teams means that not all RAPs are equal, the complexity of RAP domains means that not all *roles* are equal. For example, fires vary in severity, as well as in the value of the building at stake (e.g., due to human lives being at risk). We model the distinctions among roles by associating a *priority* with each. The priority of a role, r, maps its state (e.g., severity of fire) into a total ordering over roles: $p_r : S_r \rightarrow [0, 1]$. Unlike capabilities, priorities may be unknown to the RAPs and their proxies in advance. The job of determining the priority of a role is a specialized coordination role that could be allocated to a RAP with special skills for determining priorities of roles. For example, the fire chief may be able to provide up-to-date prioritization information on the fires, but the fire brigades themselves may not have that capability. Just as with capabilities, we can also represent the imperfect knowledge that the RAPs have with an approximation of the true priority.

4.2 Role Allocation Algorithm

Figure 3 shows pseudo-code for our new role allocation algorithm. The rest of this section presents a high-level description of the algorithm, as well as some of the intuitions that motivate its designs and that underly its operations.

New domain level roles are created by the progression of team plans. Initially, there is no RAP assigned to a newly created domain-level role. The unassigned role triggers the creation of a *role-allocation-role*, with the responsibility of assigning a RAP to the domain-level role (Line 2). By default, the proxy that creates the role-allocation role is initially responsible for performing that role[3] (Line 3).

[3]This simply prevents a recursion of role creations and is reasonable since, unlike the domain-level role, it is the proxy taking responsibility not the RAP.

After recursively calling the role-allocation algorithm for this new role-allocation role (Line 4), the proxy attempts to find a RAP capable of performing the domain-level role (Line 5). It first considers whether it could allocate the role to its RAP by invoking its adjustable-autonomy algorithm (Line 6). If its RAP is potentially capable of performing the role, the proxy engages in adjustable autonomy reasoning to decide whether it will autonomously decide to reject or accept the role on behalf of the RAP or whether it will ask the RAP to make the decision. If the role is accepted (either autonomously or by the RAP itself), the role-allocation-role terminates and the accepting RAP takes on responsibility for the domain-level role (Line 9).

Algorithm 3: AllocateRole

```
ALLOCATEROLE(r)
(1)  if no one is assigned to r
(2)      create role-allocation role, r', for r
(3)      assign myself to r'
(4)      ALLOCATEROLE(r')
(5)  else if I am assigned to r
(6)      B ← ADJUSTABLEAUTONOMY(r)
(7)      Append myself to asked(r)
(8)      if role accepted ∈ B
(9)          RAP executes role r
(10)     else
(11)         if |asked(r)| < maxAsked · |capable(r)|
(12)             Assign select(capable(r) \ asked(r)) to r
(13)         else
(14)             De-assign myself from r
(15)         ALLOCATEROLE(r)
(16) else
(17)     do nothing
```

On the other hand, if the RAP rejects the role (Line 10), the proxy's role-allocation-role requires that it attempt to find another RAP. We use the *maxAsked* parameter to control how many of the capable RAPs we will try before giving upare asked before the team "gives up" (Line 11). Notice that a single proxy will only transfer the role to one RAP, then that RAP may in turn pass it to another RAP. Hence, the *maxAsked* parameter is equivilent to the number of RAPs the role allocation role visits. The *select* procedure in Line 12 of Figure 3 represents the proxy's decision in choosing such a RAP, based on some representation of the other RAPs in its team. How much a proxy knows about other RAPs will vary. It will consider passing a role to another RAP and proxy unless it has specific knowledge that that RAP is incapable of performing the role. Instead of directly asking the other RAP itself, the proxy passes responsibility for the role-allocation-role to the other RAP's proxy. The newly responsible proxy then follows the same process as the originally responsible RAP, with one key difference. By keeping track of all of the past RAPs responsible for this role (in the *asked* set, updated in Line 7), we guarantee that the newly assigned RAP will not simply pass the role-allocation-role back to the original proxy.

A proxy may not always be able to find an appropriate RAP to which to pass a role-allocation-role or the role-allocation-role may have been transferred to more than *maxAsked* RAPs (Line 13). In such a situation, the proxy is failing at its role and attempts to reallocate the role-allocation role to a RAP who may be in a better position to do the reallocation (Line 14). Through another recursive call to our role-allocation algorithm (Line 15), we essentially move up another meta-level away from the original domain-level role. In other words, reallocating the role-allocation-role triggers yet another role, this one to reallocate the role-allocation-role (i.e., a *role-allocation role-allocation-role*). By moving up a meta-level,

the algorithm eventually transfers responsibility to a RAP who is especially capable at such role allocations.

To make the algorithm more concrete, consider the following example. Fire brigade A is fighting a fire in building 1, while fire brigade B is fighting a fire in building 2. A fire chief is at headquarters collecting information from a variety of sources. Fire brigade A notices another fire in building 3. This observation triggers a plan to fight the in building 3, which in turn triggers the role *fire fighter* within that new plan. The proxy for fire brigade A invokes the role-allocation algorithm for this new role. Following our algorithm, it automatically creates a role-allocation-role for the unassigned *fire fighter* role. The proxy decides that it should autonomously accept/reject the role for its fire brigade agent. It decides to reject the role, since the fire brigade is already engaged in fighting another fire. It decides to pass the role-allocation role to the proxy of fire brigade B. Fire brigade B's proxy also decides to act autonomously on behalf of the fire brigade in rejecting the role. Fire brigade B's proxy has no other capable RAP to pass the *fire fighter* role onto. So a reallocation of the role allocaion role is required. The fire chief is a specialist at role reallocation, so fire brigade B's proxy passes the role to the chief's proxy. The fire chief's proxy offers the role-allocation-role to the chief, who accepts it. The chief has information that building 3 contains young children while building 1 is largely empty. The chief performs its role by reassigning fire brigade A to building 3. The newly abandoned role for fighting the fire in building 1 cannot be allocated in a distributed manner, so eventually a role to do the role-allocation is created and sent to the chief, who will "hold" the role until either a fire brigade becomes available or the plan becomes irrelevant (e.g., because the fire burns out).

Clearly, this algorithm will not always find optimal allocations of RAPs to roles. We could improve the quality of the allocations by improving the decision making underlying the proxies' acceptance/rejection of roles and by improving the choice of RAP to which to pass rejected roles. However, we can potentially achieve more improvement by creating role-reallocation roles for several roles and sending all the role-reallocation roles to a capable RAP (or perhaps RAPs) for a more centralized role assignment. These role-allocating expert RAPs can work on finding improved allocations while the RAPs continue to execute their own roles.

5. TESTBED AND EXPERIMENTS

The role-allocation procedure is not a single algorithm, but rather represents a space of possible algorithms that are easily realizable through variation of its parameters. The resulting implementation of this procedure within our proxy architecture provides us with an invaluable testbed for empirical analysis of the space of role-allocation algorithms. This section presents the results of our exploration of parts of this space and the implications for RAP teams.

5.1 Role Allocation Testbed

There are a number of decision points and parameters in the role-allocation algorithm of Figure 3 that offer a system designer a dimension of control over the coordination reasoning. For example, the *maxAsked* parameter controls the number of other RAPs that should be tried before creating a role-allocation-role-allocation role. If *maxAsked*$= 0\%$, a proxy whose RAP cannot (or will not) take on the role will give up and immediately create a role-allocation-role-allocation role. If *maxAsked*$= 100\%$, the algorithm ensures that all capable RAPs are offered the role once before giving up. At the extreme, a special setting of *maxAsked*$= \infty$ means that the capable RAPs repeatedly pass the role amongst themselves (with each getting offered the role multiple times) without ever giving up. Varying *maxAsked* throughout this range produces distinct algorithms that produce different loads on role-allocation expert RAPs (e.g., a fire chief).

Furthermore, our role-allocation algorithm can take its input on RAP capabilities and role priorities in different forms, as described in Section 4.1. When a proxy autonomously decides whether to accept a role or not (as modeled by the "*accept autonomously*?" decision in Algorithm 2), it weighs its current role against the newly offered one (e.g., is the new fire higher priority?). Information required to make such a determination (e.g., priority information) may generate new roles that undergo this same role-allocation process (e.g., by triggering a "determine priority" role).

Regarding RAP capabilities, if our role-allocation uses the exact dynamic, quantitative capabilities, c_i, then the proxies can perhaps make high-quality allocations on their own, without appeal to an allocation expert RAP. We would expect that use of an approximate c_i would lead to lower-quality autonomous allocations, but would reduce the number of messages sent to update each RAP's local state. This distinction also affects the *select* procedure in the ALLOCATEROLE algorithm. Accurate capability knowledge allows the proxy's to pass the role on to the most able RAP available, rather than simply passing it on to the first one found.

Regarding role priorities, again, if our role-allocation uses the exact, quantitative information, p_r, for each role r, then the proxies can make high-quality autonomous allocations. However, to acquire this precise priority information, the proxies must typically appeal to a role-prioritization expert RAP (e.g., a fire analyst). Thus, there is a tradeoff between the amount of effort the exact information saves the allocation expert and the additional effort now burdening the prioritization expert. If the role-allocation uses the approximate, binary priority information, then the tradeoff swings to the opposite direction.

5.2 Experimental Setup

Our purpose in designing this testbed is to explore different styles of coordination among RAPs and their effects on team performance. We have presented several dimensions along which our proxy architecture can vary its coordination behavior, but given the time required for experimentation (a single run requires 45 minutes), we have performed an exploration over a subset of these dimensions, rather than a shallower breadth-first approach. In particular, we focus on two factors: the complexity of the environment and the point at which roles were allocated to people. Our aim is to gain insight into the benefits (and potential drawbacks) of giving people the opportunity to make coordination decisions. To isolate these benefits, we used a RAP team consisting of a single person (as fire chief) and a number of agent-controlled fire brigades. We have omitted robots from the team for the purposes of this particular experiment.

Along the dimension of domain complexity, we varied the number of fire brigades, with one configuration using 3 and another using 10. Along the dimension of allocating roles to people, we used three extreme parameter settings to vary the degree of control that the person has over role allocations: (i) the agents give up control as soon as the first agent rejects the role (*maxAsked* $= 0\%$), (ii) the agents give up control only after *all* of the agents have rejected the role once (*maxAsked* $= 100\%$), and (iii) the agents *never* give up control (*maxAsked* $= \infty$). We fixed every other aspect of the initial state, i.e., the fire brigades had fixed starting positions, and there were two predetermined fire ignition locations.

The fire chief interface consists of two frames. One frame shows a map of the city, displaying labeled markers for all of the fires that have been found, the positions of each fire brigade, and the location of the role each fire brigade is assigned to. The fire chief does not

# Brigades	*maxAsked*= 0%	*maxAsked*= 100%	*maxAsked*= ∞
3	58(3.56)	73(16.97)	74(0.71)
10	52(19.09)	42(14.00)	73(4.24)

Table 1: Domain-level performance scores.

have direct access to the simulation state through the simulator itself, but is instead updated according to only the messages received by the fire chief's proxy. Therefore, the fire chief may be viewing a delayed picture of the simulation's progress. The other frame displays a list of all of the role-allocation tasks that have been allocated to the fire chief. By clicking on a task, the relevant capability information about each fire brigade is shown. The right-side window lists the fire brigades' distances to the fire, their water levels, and the roles they are currently performing. The fire chief can then view this data and find an appropriate agent to fulfill the role.

We conducted tests with three different fire chiefs. Each completed several practice runs with the simulation prior to experiments in order minimize any learning effects. Each scenario was run for 100 time steps, with each step taking 30 seconds. The total data presented here represents 20 hours of run-time with a human in the loop.

5.3 Experimental Results

Table 1 shows the team's domain-level performance across each experimental configuration. The scoring function measures how much of the city was destroyed by fire, with higher scores representing worse performance. The table shows the mean scores achieved, with the standard deviations in parentheses. Examining our two dimensions of interest, we can first compare the two rows to examine the effect of increasing the complexity of the coordination problem. In this case, increasing the number of fire brigades improves performance, as one might expect when adding resources while keeping the number of initial tasks fixed.

However, we can dig a little deeper and examine the effect of increasing complexity on the fire chief's performance. In the simpler configuration, asking the fire chief earlier (i.e., *maxAsked*= 0) improves performance, as the team gets a head start on exploiting the person's capabilities. On the other hand, in the more complex configuration, asking the fire chief earlier has the opposite effect. To better understand the effect of varying the point at which we assign roles to people, Table 2 presents some of the other statistics we gathered from these runs (mean values, with standard deviations in parentheses). With 3 brigades, if we count the mean number of roles taken on by the fire chief, we see that it stays roughly the same (401 vs. 407) across the two *maxAsked* settings. In this case, asking the fire chief sooner, allows the team to exploit the person's capabilities earlier, without much increase in his/her workload. On the other hand, with 10 brigades, the fire chief's mean role count increases from 563 to 716, so although the proxies ask the fire chief sooner, they are imposing a significant increase in the person's workload. Judging by the decreased average score in the bottom row of Table 1, the increased workload more than offsets the earlier exploitation of the person's capabilities. Thus, our experiments provide some evidence that increasing domain-level scale has significant consequences on the appropriate style of interaction with human team members.

Regardless of the variation of human behavior across scale, the data demonstrates that exploiting human capabilities does, in fact, improve overall team performance. We see this most clearly by examining the rightmost column of Table 1, which represents the results when the agents make all of the decisions. These scores are significantly worse than the leftmost data column, where the person is handed role-allocation roles immediately. Thus, the ability of our role-allocation algorithm to exploit the special coordination capabilities of people has provided a dramatic improvement in the performance of our RAP team.

# Brigs.	*max Asked*	Domain Roles	Fire Chief Roles	Tasks Performed	% Tasks Performed
3	0%	116 (7.12)	401 (51.81)	27 (6.55)	23.29 (6.51)
	100%	146 (33.94)	407 (54.45)	24 (6.36)	16.02 (0.63)
10	0%	103 (38.18)	864 (79.90)	67 (2.83)	14.49 (2.13)
	100%	98 (42.40)	563 (182.95)	41 (8.38)	48.06 (19.32)

Table 2: Role and fire-chief task metrics.

Fire Chief	Score	Tasks Performed	% Performed
A	0.61	25	28%
B	0.59	40	56%
C	0.31	42	32%

Table 3: Statistics for each Fire Chief.

We can observe the heterogeneity introduced by people by clustering our statistics by person rather than by configuration. Each row in Table 3 represents the mean statistics of one of our three different fire chiefs. The "Tasks Performed" column counts the number of firefighting allocations performed by the fire chief, while the "% Performed" column measures that count against the number of total firefighting allocations assigned to the fire chief by the proxy architecture. Given the small sample size, we cannot draw any conclusions about a person's expected behavior. On the other hand, it *is* clear that we can expect a great deal of variance in behavior. For example, although fire chiefs A and B achieve roughly similar mean scores, they do so in very different ways. In fact, our proxies can expect fire chief A to be half as likely as fire chief B to respond to a task request. On the other hand, fire chief C is about equally likely as A to respond, and C performs roughly the same number of tasks as B, yet C achieves only half the score as the other two. Thus, it appears unlikely that we can easily classify people's capabilities, since, for even the relatively few dimensions measured here, our human fire chiefs show no generalizable characteristics.

6. RELATED WORK

Proxy-based integration architectures are not a new concept, however no previous architecture has been explicitly designed to have robots, agents and people in the same team. Furthermore, all previous arhitectures aspire to give all coordination responsibilities to the system, thus preventing humans from making coordination decisions. Jennings's GRATE* [7] uses a teamwork module, implementing a model of cooperation based on the joint intentions framework. Each agent has its own *cooperation level* module that negotiates involvement in a joint task and maintains information about its own and other agents' involvement in joint goals. Jones [8], Fong [21], Kortenkamp[11] and others have worked on improving collaboration between groups of robots and a single person, though these approaches to robotics teams have not explicitly used proxies. The Electric Elves project was the first human-agent collaboration architecture to include both proxies and adjustable autonomy[2]. COLLAGEN [18] uses a proxy architecture for collaboration between a single agent and user. Payne et al[23] illustrate how variance in an agent's interaction style with humans affects performance in domain tasks. Tidhar [25] used the term "team-oriented programming" to describe a conceptual framework for specifying team behaviors based on mutual beliefs and joint plans, coupled with organizational structures. The framework also addressed the

issue of team selection [24] by matching "skills" required for executing a team plan against agents that have those skills.

Role allocation has been considered by a number of authors using a range of different techniques. For example, there are role allocation approaches for doing an allocation of RAPs to a static set of roles based on capability analysis [24], combinatorial auctions [6] and Beliefs Desires and Intentions [19]. Role reallocation has also been considered, e.g., team reorganization based on *intends.that* in the SharedPlans approach[5] or *critical role failures* in STEAM [22]. Role allocation for teams of robots has been considered by several authors including Gerkey [4], Parker [14] and Ostergaard[13]. However, there has been no work that allows both a distributed algorithm and a person to participate in the role allocation.

7. LESSONS LEARNED

While the goal of building completely autonomous agents is an admirable one, we believe that the pendulum has swung too far towards autonomy. The ability to leverage the skills of humans via adjustable autonomy is not the result of immature agent technology but a fundamental requirement for any agent that is to cohabit an environment with people. Without adjustable autonomy agents miss out on the possibility that willing and able people can provide valuable input that improves the performance of the system. In the specific case of proxy-based integration architectures for RAP teams, architectures where proxies have all responsibility for performing coordination precludes the possibility that willing and able RAPs can provide input that will improve the team's coordination. We have developed a novel proxy-based integration architecture where coordination is performed by both RAPs and proxies, synergistically utilzing their respective strengths for the benefit of the team. The key idea is to treat coordination tasks the same way as domain level tasks, that is by representing them explicitly as roles that can be allocated to any willing and able entity. Through the use of adjustable autonomy, the proxies flexibly share coordination responsibility with their RAPs to the benefit of the team. Experiments with the new proxies in an urban disaster recovery domain showed that giving people the opportunity to assist in the coordination did improve the performance of the team.

Acknowledgments

This research was supported by DARPA award no.F30602-01-2-0583. We would like to thank Dylan Schmorrow at DARPA and Allen Sears at CNRI. We also thank our colleagues Guarav Sukhatme, Sameera Poduri and Dennis Wolf for their robotics expertise and Praveen Paruchuri for testing.

8. REFERENCES

[1] J. Casper and R. Murphy. Workflow study on human-robot interaction in usar. In *International Conference on Robotics and Automation*, pages 1997–2003, 2002.

[2] Hans Chalupsky, Yolanda Gil, Craig A. Knoblock, Kristina Lerman, Jean Oh, David V. Pynadath, Thomas A. Russ, and Milind Tambe. Electric Elves: Agent technology for supporting human organizations. *AI Magazine*, 23(2):11–24, 2002.

[3] Philip R. Cohen and Hector J. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.

[4] Brian P. Gerkey and Maja J Mataric. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation, Special Issue on Multi-robot Systems*, 2002.

[5] B. Grosz and S. Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–358, 1996.

[6] Luke Huhnsberger and Barbara Grosz. A combinatorial auction for collaborative planning. In *Proceedings of the International Conference on MultiAgent Systems*, 2000.

[7] N. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75, 1995.

[8] Henry L. Jones, Stephen M. Rock, Dennis Burns, and Steve Morris. Autonomous robots in swat applications: Research, design, and operations challenges. In *AUVSI '02*, 2002.

[9] Hiroaki Kitano, Satoshi Tadokoro, Itsuki Noda, Hitoshi Matsubara, Tomoichi Takahashi, Atsushi Shinjoh, and Susumu Shimada. Robocup rescue: Searh and rescue in large-scale disasters as a domain for autonomous agents research. In *Proc. 1999 IEEE Intl. Conf. on Systems, Man and Cybernetics*, volume VI, pages 739–743, Tokyo, October 1999.

[10] D. Kortenkamp, G. Dorias, and K. Myers, editors. *Proceedings of IJCAI99 Workshop on Adjustable Autonomy Systems*, August 1999.

[11] D. Kortenkamp, D. Schreckenghost, and C. Martin. User interaction with multi-robot systems. In *Proceedings of Workshop on Multi-Robot Systems*, 2002.

[12] P. J. Modi, H. Jung, W. Shen, M. Tambe, and S. Kulkarni. A dynamic distributed constraint satisfaction approach to resource allocation. In *Proc of Constraint Programming*, 2001.

[13] E. Ostergaard, M. Mataric, and G. Sukhatme. Multi-robot task allocation in the light of uncertainty. In *IEEE International Conference on Robotics and Automation*, pages 3002–3007, 2002.

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

[15] David Pynadath and Milind Tambe. Multiagent teamwork: Analyzing the optimality and complexity of key theories and models. In *First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'02)*, 2002.

[16] David V. Pynadath and Milind Tambe. 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*, page to appear, 2002.

[17] D.V. Pynadath, M. Tambe, N. Chauvat, and L. Cavedon. Toward team-oriented programming. In *Intelligent Agents VI: Agent Theories, Architectures, and Languages*, pages 233–247, 1999.

[18] C. Rich and C. Sidner. COLLAGEN: When agents collaborate with people. In *Proceedings of the International Conference on Autonomous Agents (Agents'97)*, 1997.

[19] K. Seow and K. How. Collaborative assignment: A multiagent negotiation approach using bdi concepts. In *Proceedings of AAMAS'02*, pages 256–263, 2002.

[20] P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low bandwidth communication for real-time stategic teamwork. *Artificial Intelligence*, 110(2):241–273, June 1999.

[21] C. Thorpe T. Fong and C. Baur. Advanced interfaces for vehicle teleoperation: collaborative control, sensor fusion displays, and web-based tools. In *Vehicle Teleoperation Interfaces Workshop, IEEE International Conference on Robotics and Automation*, San Fransisco, CA, April 2000.

[22] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[23] Katia Sycara Terry Payne and Michael Lewis. Varying the user interaction within multiagent systems. In *Agents'00*, pages 412–418, 2000.

[24] G. Tidhar, A.S. Rao, and E.A. Sonenberg. Guided team selection. In *Proceedings of the Second International Conference on Multi-Agent Systems*, 1996.

[25] Gil Tidhar. Team-oriented programming: Preliminary report. Technical Report 41, Australian Artificial Intelligence Institute, 1993.