CAST: Collaborative Agents for Simulating Teamwork

John Yen, Jianwen Yin, Thomas R. Ioerger,
Michael S. Miller, Dianxiang Xu, and Richard A. Volz
Department of Computer Science
H. R. Bright Building
Texas A&M University
College Station, TX 77843-3112, USA
{yen, jianweny, ioerger, mmiller, xudian, volz}@cs.tamu.edu

Abstract

Psychological studies on teamwork have shown that an effective team often can anticipate information needs of teammates based on a shared mental model. Existing multi-agent models for teamwork are limited in their ability to support proactive information exchange among teammates. To address this issue, we have developed and implemented a multi-agent architecture called CAST that simulates teamwork and supports proactive information exchange in a dynamic environment. We present a formal model for proactive information exchange. Knowledge regarding the structure and process of a team is described in a language called MALLET. Beliefs about shared team processes and their states are represented using Petri Nets. Based on this model, CAST agents offer information proactively to those who might need it using an algorithm called DIARG. Empirical evaluations using a multi-agent synthetic testbed application indicate that CAST enhances the effectiveness of teamwork among agents without sacrificing a high cost for communications.

1 Introduction

Teamwork has been the focus of a great deal of research, spanning diverse disciplines from business management to psychology [Ilgen et al., 1993]. There are many different types of teams, from those that are hierarchical to those that are more egalitarian. Some teams have fixed, clearly-defined roles, while others allow for dynamic re-allocation of tasks and responsibilities on the fly. Measuring the performance of a team can involve both "outcome" as well as "process" measures [Cannon-Bowers and Salas, 1997].

Several computational models of teamwork have been developed for producing cooperative behavior among intelligent agents. The fundamental aspect of a team that distinguishes them from just a group of interacting agents is that they share common goals. In the BDI framework [Rao and Georgeff, 1991; Wooldridge and Jennings, 1995], the mental state of having a team goal has been characterized in terms of *joint intentions* [Cohen and Levesque, 1991, Jennings, 1995]. Tambe [Tambe, 1997] and his STEAM group have shown how these mental states can be established and maintained through communication protocols. Another framework for modeling teams is through *SharedPlans*, via intentions to do certain steps together [Grosz and Kraus, 1996]. These approaches have been shown to be effective for simulating teamwork in a wide range of agent-only environments [Jones et al, 1999; Tidhar et al., 1998; Stone and Veloso, 1999].

However, these existing multi-agent teamwork models have not been designed for supporting mixed human/agent teams. Having humans in the loop places an additional constraint on the agents, such that they must interact with teammates in a natural way, e.g. by exchanging only the most important information necessary for coordination, without excessive or redundant communication. Efficient teamwork relies heavily on information sharing, especially in dynamic environments, but it must be done judiciously not to overwhelm the human participants with message passing. The key is to try to supply only the most *relevant* information, and this requires reasoning about their goals and responsibilities on the team.

Motivated by this observation, we have developed CAST (Collaborative Agents for Simulating Teamwork), a multi-agent architecture that simulates and supports teamwork. While our ultimate goal involves supporting both humans and agents, in this paper we will focus on reducing communication with software agents (in preparation for inclusion of humans) as a first step. Our architecture provides a mechanism for building virtual teams using software agents. We will first give an overview about the goals and issues addressed by CAST. A formal

foundation for proactive information exchange in CAST is then introduced. This is followed by a description of the major components and features of the CAST architecture. An empirical evaluation is used to assess the effectiveness of teamwork simulated in CAST. Finally, we summarize the main contribution of the work.

2 CAST Overview

CAST is designed to achieve two goals. First, it aims to model effective teamwork by capturing both *team structures and teamwork processes*. A well-defined team structure is based on specifying pre-defined roles and the responsibilities associated with them. A well-defined teamwork process specifies goals, strategies, and plans for accomplishing the team's goal. The common prior knowledge about the structure and the process of the team enables members of the team to develop an "overlapping shared mental model," which is the source for team members to reason about the states and the needs of others.

One of the major criticisms of other approaches toward the goal above has been that they lack flexibility for dealing with dynamic environments. Hence, another equally important goal is to enable agents in a team to have *flexibility* in adapting to changes in the environment. More specifically, assignment of responsibilities to suitable agents needs to be adapted to the current state of the world and the team.

While both goals are desirable, they conflict with each other. The emphasis on predefined team structures and processes often reduces the flexibility of the team, while maximizing the flexibility of the team usually requires making shared, redundant, and hence ambiguous role assignments. To balance these two conflicting goals, we need a practical and flexible computational framework for representing and reasoning about the overlapping shared mental models among teammates. Such a framework would enable an agent to dynamically reason about the status of the entire team and adapt its behavior accordingly.

Representing a shared mental model is a challenging problem. It can be viewed as a kind of belief reasoning, which is generally intractable [Halpern and Moses, 1992]. Moreover, the content of the shared mental model is quite broad, ranging from shared domain knowledge to a common relevant picture of the situation. We tackle this problem by focusing on two specific uses of the shared mental model: making teamwork efficient through *anticipating* the actions and expectations of others (e.g. by knowing others' roles, capabilities, and commitments), and by *information exchange* (knowing who to ask for information, or providing information proactively just when it is needed by someone else to accomplish their task). To avoid issues of computational complexity with belief reasoning (e.g. higher-order modal logics), we use Petri Nets as an approximate finite and computable model of mental states.

The Petri Net is a natural representation for parallel action and synchronization in a multi-agent world [Sowa, 2000]. Transitions can represent actions, with input places corresponding to pre-conditions and output places corresponding to effects. We extend the standard (colored) Petri Net formalism with special kinds of places called *control nodes* and *belief nodes*. Control nodes represent the belief an agent has about the current goals and activities of others in the team. Belief nodes represent the belief an agent has about the world, when coupled with a unification-based theorem-prover, can represent first-order knowledge, including dynamic facts and inferences about the world. In addition to serving as the shared mental model, Petri Nets also play the dual role of monitoring and tracking the execution of team plans.

CAST generates the Petri Net-based representation using two kinds of knowledge: 1) team structures (roles and responsibilities), and 2) teamwork process knowledge (e.g., individual plans, team plans). They are described in a knowledge representation language called MALLET (a Multi-Agent Logic-based Language for Encoding Teamwork). A Petri Net for a given team member represents both the background knowledge for their individual responsibilities (e.g. goals, operators), and how their role is integrated with the rest of the team.

Based on the shared mental model, the CAST kernel enables CAST agents to decide on the fly how to accomplish desired goals, how to select responsibilities to commit to or delegate, how to proactively assist others in the team, and how to effectively communicate within the team. This is achieved by dynamic role selection and proactive information change. We will give a more detailed description in later sections.

3 Formal Foundation of Proactive Information Flows

In this section, we discuss the formal foundations underlying the type of information exchange addressed in CAST. In particular, we think it is important to specify, *under ideal conditions*, what information should be exchanged between whom, and at what time. The purpose of information exchange must be oriented toward improving the efficiency or performance of a team, but otherwise is desired to be kept to a minimum to avoid the cost of communications overhead (however this is defined in the domain). In fact, a quantitative utility function that incorporates such costs is actually used in STEAM [Tambe, 1997] to help evaluate tradeoffs and decide when to communicate within a team.

In CAST, we take a different approach to optimizing information exchange by defining narrow criteria for the exchange of only the most critical information. Specifically, we want agents who know some fact to communicate it to exactly those teammates who need the information in the present context to carry out their goals,

and who furthermore (probably) do not already know it. Clearly, this might involve some complex reasoning about beliefs and goals, as well as tracking the state of other agents. However, to the degree that some of these inferences can be approximated, the team members can make intelligent decisions to selectively choose their interactions with one another.

We start by defining a simple belief language and model theory to be able to talk about the mental states of various. We model beliefs using a modal operator BEL, e.g. `(BEL bill (have joe hammer))`, with the usual possible worlds semantics [Cohen & Levesque, 1990]. We need to be able to talk about 'pieces' of information, which in this present context refer *syntactically* to sentences, but *semantically* are equivalent to constraints over possible worlds, i.e. those worlds satisfying the expression. Goals, however, refer to specific steps in plans in MALLET, to which agents can make commitments.

Using this framework, we can formally characterize the (normative) conditions under which information exchange should take place. Information *I* should be sent from one agent *A* to another agent *B* when: 1) agent *A* knows the truth-value of *I*, 2) agent *A* believes that agent *B* does not currently know *I*, and 3) *B* has a current goal *G*, the achievement of which depends on knowing *I*, i.e. if *B* does not believe *I*, then it will never be able to accomplish it's goal, but if it knew *I*, it would be able to:

$$(\text{BEL A I}) \wedge (\text{BEL A } \neg(\text{BEL B I})) \wedge$$
$$(\text{BEL A (GOAL B G)}) \wedge$$
$$[\neg(\text{BEL B I}) \rightarrow \Box\neg(\text{DONE B G})] \wedge$$
$$[(\text{BEL B I}) \rightarrow \neg\Box\neg(\text{DONE B G})]$$
$$\rightarrow (\text{GOAL A (Inform B I)})$$

where $\Box$ is the temporal operator for 'always'. In CAST, we use the pre-conditions of operators to determine the information *I* that an agent needs to know to achieve its goals. Note, the communication is suppressed only when (*A* believes that) *B* already believes *I* is true; but if *B* does not have a belief about the truth value of *I* at all, or *B* *incorrectly* believes it is false, then the message will be sent. For simplicity, we assume that all agents share common (and correct) knowledge of the team, e.g. team goals and plans, roles and responsibilities, operator pre-conditions, etc.

Clearly this approach requires *A* to monitor *B*'s mental state, both in terms of what his beliefs and goals (commitments) are. This might be easy to do in a highly observable environment, but might require extensive communication in other cases (e.g. verbal updates of what each teammate is currently doing). Alternatively, agents might use a probabilistic mechanism like Bayesian reasoning to infer the likely states of their teammates, based on observations of the effects of their actions reflected in the environment. Regardless of how difficult this state estimation or tracking might be to implement, the definition above describes the *ideal* conditions under which one would want to communicate. As much as possible, we want to restrict communication to cases where it can be inferred to be useful, which is what the DIARG algorithm below is designed to approximate.

4 Specifying Team Knowledge in MALLET

In this section, we briefly describe MALLET, the team knowledge representation language in CAST. MALLET provides descriptors for encoding knowledge about operators and plans of individuals and the team, as well as definitions of roles and responsibilities on the team. An important ontological commitment in MALLET is that *roles* are fundamentally treated as *references to specific steps in team plans*, to which certain agents on the team can externally be assigned. That is, the meaning of a role is defined by a certain step in a team plan that is dedicated for the agents playing that role to carry out.

MALLET syntax is based loosely on LISP, in the sense of using s-expressions and prefix notation. Variables are indicated with '?' prefix. Operators are simply names of atomic actions that can be taken in the environment. Operators may list a set of pre-conditions and/or post-conditions, each as a conjunction of literals (first-order predicates or their negations). Here is an example for climbing over a pit:

```
(operator climb-over (?pit)
(pre-cond (have-ladder)))
```

There are two types of operators: *individual* operators and *team* operators. Individual operators are assumed to be executed by only one agent at a time. However, team operators have the possibility of being invoked on a *set* of agents (e.g. those playing a given role). How the agents handle the team operators depends on what sub-type it has. We have identified three modes of operator-sharing:

- AND operators, which require simultaneous action by all the agents involved
- XOR operators, which require at most one agent to act (mutual exclusion, e.g. to avoid conflicts)
- OR operators, which can be execute by any of the agents (possibly >1) without conflict

Hence team operators contain an extra component that defines the "share-type." For example, an operator for lifting a heavy object might be written as:

```
(t-operator lift-heavy (?x)
   (share-type AND)
   (pre-cond (movable ?x)))
   (effect (holding ?x)))
```

Plans in MALLET are essentially designed to describe *processes*. Processes consist of invocations of atomic actions, or arbitrary combinations using various constructs such as sequential, parallel, contingent, or iterative. The syntax of processes can be defined recursively according to five constructs, with obvious intuitive meanings:

```
(seq P Q), (par P Q), (if (cond C) P
[Q]), (while (cond C) P), (do T)
```

where P and Q are sub-processes, C is condition (conjunction), and T is an operator or plan instantiation (with arguments). These constructs for describing complex processes can be given a semantics in various formalisms such as dynamic logic [Harel, 1984].

Individual plans have pre-conditions and effects, and also a process description. Team plans are similar to individual plans, but they have two extra features related to assignment of roles. First, we have declarations of role variables of the form `(role <role-name> <role-variable> <constraints>)`, where the constraints are a conjunctive set of conditions that put restriction on the role variable, which candidates must satisfy. For example, the team plan below requires two agents, one with the carrier role, the other with the fighter role. The fighter agent has an additional constraint, i.e. satisfying the predicate *closestToWumpus*.

```
(t-plan explore-cave ()
 (role carrier ?ca)
 (role fighter ?fi ((closestToWumpus ?fi) )
 (process (seq
          (do ?ca (find-wumpus))
          (do ?fi (moveto-wumpus))
          (do ?fi (kill-wumpus)) ) ) )
```

The general idea is that an appropriate agent meeting the constraints will be selected from the set of those assigned to play the role, and bound to the role variable within the plan. These role variables can then be used within processes to specify which agents will do certain steps (operators or sub-plans). An important implication of allowing constraints in the role specification is that it introduces flexibility to the teamwork process because the selection of such roles needs to be made dynamically at run time to assure that the constraint is satisfied. We will elaborate on the role selection scheme in the next section.

Finally, MALLET also provides simple descriptors for defining the members on the team, the roles they play, and their capabilities and responsibilities. While capabilities are treated simplistically (as static associations between agents and operators), the meaning of a responsibility is more interesting. It is similar to a role, in the sense that it defines agents who are supposed to do certain actions, but they are not tied to specific steps in the context of specific plans. Instead, the responsibility of an agent for an operator means that, whenever the action needs to be done at any time, the agent knows that it should act or at least coordinate with others who share the responsibility.

MALLET descriptions are converted into Petri Nets on which the agent's reasoning algorithm operate. A Petri-Net generation algorithm constructs transitions for each atomic operator in a team plan, connects them with control nodes, and links their inputs and outputs to appropriate belief nodes based on pre- and post-conditions. Since plans can be hierarchical, the sub-plans are expanded by calling the Petri-Net generation algorithm recursively, and linked into the main Petri Net through control nodes such that all the action nodes ground out in operators. Thus for each team goal, there is a single Petri Net that describes its plan for solving it. By placing tokens on the topologically-first node(s) of the Petri Net, and moving them forward whenever steps are completed, agents can keep track of the progress of the team. Though not every agent is involved in or responsible for every step, this Petri-Net model of the overall team plan forms a common understanding of the team's goals and process, which they use to determine how their individual actions fit together, and is thus a primary constituent of their *shared mental model*.

The knowledge compiler also performs a static information-flow analysis [Yin et al., 2000] for the online DIARG algorithm. An information-flow relation *I* is defined as a 3-tuple, <*Info,needers,providers*>, where *Info* is the predicate name together with 0 or more arguments, *needers* is a list of agents who might need to know such information, and *providers* is a list of agents who might know the information. We determine the needers of information by analyzing the pre-conditions of operators for which each agent might be responsible. And we determine the potential providers of information by analyzing the post-conditions of operators for which the agents might be responsible. In particular, if *P* is a post-condition of operator *O*, then we assume that an agent will know *P* after executing *O*, since we can expect agents to know the (direct) consequences of their own actions.

5 CAST Agent Kernel

The CAST kernel refers to a set of algorithms that CAST agents use to determine what actions and communications they will take at each time step. CAST agents execute a standard *sense/decide/act* loop. During the *sense* phase, they make queries to the simulation server to update their knowledge of the state of the world. Agents also check a queue for messages from other agents at this time. During the *decide* phase, each agent examines the Petri-Net representation of the team plan to see if there are any pending actions for which it is responsible. In cases of ambiguity, the agents might have to communicate in order to determine who will take the action and when. Before finally taking actions, the agents attempt to determine if there are any interactions they can initiate. For proactive information exchange, this is accomplished by DIARG.

CAST also uses a back-chaining theorem-prover called JARE (also implemented in Java) for making inferences using domain knowledge written in the form of a separate Horn-clause knowledge base. JARE is used to determine the truth-value of conditions or constraints that need to be evaluated in interpreting MALLET expressions at run-time.

5.1 Dynamic Role Selection

The algorithm for Dynamic Role Selection (DRS) is used to support reasoning about role assignments and to generate

the necessary interactions among agents to correctly execute the team plan with appropriate actors for each step. It has the following main steps:

```
DRS(step)
1   let A be the set of agents potentially
    involved in the step, A=InvolvedAgents(step)
2   remove from A any agents that are incapable of
    taking the action
3   remove from A any agents that do not satisfy
    the role constraints, if defined for that step
4   if A is empty, do nothing
5   if |A|=1 and member(self,A), do(step)
6   else
    6a. if step is an AND operator,
            synchronize(step)
    6b. if step is an OR operator,
            attempt-in-parallel(step)
    6c. if step is an XOR operator,
            disambiguate(step)
```

For each active step in the team plan (nodes marked with a token in the process net), DRS starts by determining the set of agents that could be involved. This is done by considering several different ways of assignment, in a particular order of precedence:

```
InvolvedAgents(step)
   if step is of the form do(agent,action),
        then return {agent}
   if step is of the form do(role,action),
        then return {Ai} forall agents that were
        assigned to play that role,
        Ai∈RoleSet(role)
    if step refers to an operator op for which
        some agent has been assigned
        responsibility, then return {Ai} forall
        agents such that Resp(Ai,op)
   else return all the agents on the team,
        {Ai} forall agents such that Ai∈team
```

Assignments of specific agents are considered first, and then role specifications, where several agents may be assigned to the role for a given step in the plan. If no agents or roles are directly assigned, then the search for involved agents expands to considering any agents that might be responsible in general for such actions. Finally, if it is still undefined who should perform a step, then all the agents in the team are included, since this is ultimately a step in a plan to which they are all jointly committed.

After determining the set of agents potentially involved in the action, those agents that are unacceptable are filtered out. For example, if an agent is incapable of performing the action, then it is removed from the set. In addition, if any constraints were associated with the role definition, then only those agents that satisfy the constraints are retained. This is accomplished by making a query to the agent's knowledge base that is constructed from the conjunctive conditions of the constraint by substituting any variables that are bound in the current scope, plus replacing the agent/role variable with the identity of each candidate. Those agents that do not satisfy the query are removed from the set of involved agents. Other factors could also be considered at this stage, such as removing agents from consideration whose workload is too high.

After determining the set of involved agents, steps 4-6 are used by the agents to decide what to do based on the size of this set. If the set is empty, then there is simply nothing to do; the team must wait until an acceptable agent becomes available. If there is a single (unique) agent involved in the step, then the agent may act right away, without any coordination. In these cases, CAST agents are intelligent enough to act on their own. This illustrates one of the sources of efficiency in teamwork gained from *a priori* role assignments - agents can sometimes figure out what to do on their own. If there are multiple agents involved in the step (line 6 in the algorithm), then each agent's response depends on the type of operator it is. If it is an AND operator, then all the involved agents must act simultaneously. This synchronization can be accomplished by broadcasting a *ready* signal and waiting until they have heard the same from all the other agents. If the operator is an OR operator, then any of the agents may take the action. Hence they all commit to trying to carry it out. Several might actually succeed (independently) in performing the act. However, not be too wasteful, we require agents to broadcast a *success* message when they have completed the step, at which point the other agents involved are permitted to drop their commitment (this is what `attempt-in-parallel` means in step 6b of the DRS algorithm above). Finally, if the step refers to an XOR operator, only one of the agents involved may take the action, otherwise some interference might occur. Therefore, agents must agree amongst themselves who will take responsibility for the action. This disambiguation can be accomplished through a variety of protocols, such as first-come-first-served, but they all require communication. Hence agents exchange messages to select a delegate, and then this agent make a commitment to do the action and the others do not have to.

While the current algorithm for Dynamic Role Selection produces flexible teamwork that is adaptive to specific situations, it does not handle all possible situations. For example, if agents could die, lose their connection to the team, or become incapable dynamically within the environment, then the team might need to react to these changes and adjust its operation. The possibilities range on a spectrum of complexity from using redundant role assignments for providing backup behavior and load-balancing, to dynamic re-configuration of the whole team. These features could be added to the DRS algorithm, but are left for future work.

5.2 DIARG algorithm

The DIARG algorithm (Dynamic Inter-Agent Rule Generator), an extension of IARG [Yin et al., 2000]), is responsible for identifying opportunities for proactive information exchange. Instead of sending information to all

possible needers and asking all possible providers for information, agents with DIARG can keep track of the teamwork status and only send information to whom it is relevant in the current context, and ask those agent who might know the information at the moment. This makes the DIARG algorithm more dynamic. DIARG is run by each agent during each cycle, independent of the other decision-making activities, and could result in agents sending additional messages to one another. In particular, agents attempt to identify pieces of information that other agents might need to know and inform them via TELL operations. In addition, if agents need some information and they can figure out who might know it, they can generate ASK operations. These inferences are accomplished partly by examining the information-flow relations, coupled with analysis of the current state of the team.

```
ProactiveTell()
  If I is a newly-sensed piece of information,
      or I is a post-condition of the last action
      P taken by self, then
    if I is not in the knowledge base,
      assert I into knowledge base
    for each information-flow
        <predicate, needers, providers>
      if I matches predicate name and self is
          included in the providers, then
        for each agent x in the needers,
            if agent x plays a role in an active
            step,
            TELL(x,I)

ActiveAsk()
   for each active step s in the plan in which
   self is involved
      let o be the operator to which s refers
      for each pre-condition I of s
        if not(know(self, I)), then
          let Info-flow = <predicate, needers,
          providers> be the information flow in
          which I matches predicate name
          select the agent y from providers
          (active agents first),
          do ASK(y,I)
```

These algorithms are designed to generate inter-agent communications based on approximation of the criteria set forth in Section 3 for ideal conditions under which it is desirable to exchange information. In principle, we want agents to communicate only when it is likely to be useful, to minimize network traffic (use of bandwidth). The formal criteria are difficult to achieve in practice, particularly due to the need for belief reasoning. While we do not model the complete belief state of other agents, our static analysis of information flow identifies agents which might know or need to know certain information based on their involvement in the team plan (i.e. through actions for which they might be responsible). When some information changes, for example due to the action of an agent, and there is an effect of the action that others cannot observe, or when an agent observes some new information, then it might want to inform the others (proactively). The sending of these messages is restricted to those agents who conceivably might need to know the information. Whether or not an agent *really* needs to know a piece of information also depends on whether it supports one of their current goals, which we predict from active nodes in the Petri Net.

However, these algorithms cannot guarantee perfectly optimal information flow. For example, it depends on what they can observe and what they can infer. Also, while we assume that agents are never forgetful, information might change dynamically (non-deterministically) at different rates in different environments, and not all agents might be able to observe changes in all information, making the problem of maintaining consistent and correct distributed knowledge in a team very difficult in general [Singhal and Zyda, 1999]. However, the DIARG algorithm is implemented at an appropriate level for supporting interactions in mixed human-agent teams, since it does not require direct access to (or simulation of) the mental state of other team members, and is able to derive potentially useful information flows based only on analysis of common (shared) knowledge of the team plan, individuals' roles, and the current state of progress, which can be assumed to be monitored by all the members of the team.

6 A Testbed Application

We have constructed a testbed application based on a multi-agent extension of the Wumpus World [Russell and Norvig, 1995]. These agents can act as part of a team by playing the role of a fighter, to shoot the wumpus, or a carrier, to carry the gold they find. To generate the need for information flow between team members, the two types of roles have different sensing capabilities. While both of them can sense a stench (from a wumpus), a breeze (from pits), and glitter (from gold), only the carrier can pin-point the exact location of the wumpus when it is in an adjacent room in the cave. These experiments use a team plan and individual plans, along with other related teamwork knowledge, to find and kill multiple wumpuses and collect gold. Agents can communicate in three ways: (1) proactive tell, (2) broadcast information, and (3) broadcast control tokens for coordination purposes.

Table 1. Different teams used in experiments

Experiment	Team	#of Carriers	#of Fighters	Team work	Communication	DRS
1	A	1	1	Yes	Proactive Info Exchange	No
	B	1	1	Yes	Broadcast Every New Info	No
	C	1	1	No	No	No
2	D	1	3	Yes	Proactive Info Exchange	No
	E	1	3	Yes	Proactive Info Exchange	Yes

In order to evaluate the effectiveness of different features supported by CAST - shared mental models, proactive information exchange, and dynamic role selection - we have devised two sets of experiments and five multi-agent teams for comparison. The differences among the five teams are listed in Table 1. Experiment 1 is run on Team A, Team B, and Team C to show the benefit of using shared mental models and proactive information exchange. Experiment 2 is run on Team D and Team E to show the benefit of dynamic role selection.

In Experiment 1, Teams A and B both use teamwork, whereas C does not. Furthermore, Team A uses proactive information exchange (the DIARG algorithm is turned on), whereas agents in Team B just broadcast each new piece of information indiscriminately to all the members on the team. Agents in Teams A and B use a MALLET specification of a team plan that requires the carrier to first find the wumpus, then navigate a fighter to a room adjacent to the wumpus and shoot the wumpus. Agents in Team C merely wander around randomly, independently looking for wumpuses to shoot and gold to pick up. To make the comparison fair, agents in Team A and Team B also use individual plans in MALLET to perform this wandering behavior when not busy. So there is teamwork based on the shared mental model of the team plan for agents in Team A and Team B, but in Team C, there is no communication and no teamwork. Experiment 2 is meant to show the benefit of dynamic role selection. Team D is similar to team A except that Team D has more fighters; however, Team D does not use dynamic role selection. Team E implements the dynamic role selection protocol for each of its agents. All five teams use the same knowledge base (i.e. JARE rules) for reasoning about the environment and for determining the priority of actions.

Table 2. Comparison of performance for teams that differ in teamwork and information exchange.

Team	Static	V1	V2	V3	V4	V5
A	Average	3.6	7.35	2.6	1	2.2
	Std. Dev.	0.82	3.79	1.35	0	0.62
B	Average	3.5	6.55	2.55	138	0
	Std. Dev.	0.76	3.52	1.32	12.7	0
C	Average	1.6	7.85	2.2	0	0
	Std. Dev.	0.88	5.58	1.94	0	0

V1: # Wumpuses Killed V2: # of Arrows Used
V3: # of Gold Found V4: # of Messages Broadcast
V5: # of Proactive Information Exchanges

Experiment 1 was performed on 20 randomly generated maps for a world (cave) with 10 by 10 cells (rooms), 5 wumpuses, 2 pits, and 10 piles of gold. Each team is allowed to operate for a fixed number of total actions. The performance of each team for each case is measured by (1) the number of wumpuses killed, (2) the number of arrows used, (3) the amount of gold gathered (4) the number of messages broadcast, and (5) the number of proactive tell messages. The average and standard deviation of the experiments are summarized in Table 2.

As shown in the Table 2, Team A and Team B achieve comparable performance, because they use an identical shared mental model specified in MALLET. However, Team B has a much higher communication overhead (V4). Team A and Team B both have better performance in terms of number of wumpuses killed, because there is teamwork among agents in Team A and Team B. Whenever the carrier in Team A and Team B finds any wumpus, it will tell the fighter to go kill it, which gives the whole team a better chance to kill more wumpuses. However, picking up gold is an individual activity, so from the table, we can not see much difference among the three teams. One more advantage of Team A is that the fighter only needs one arrow to shoot a wumpus with known location. This is achieved because the carrier agent tells the fighter agent proactively about the location of the wumpus, using DIARG to infer that the fighter needs the information to navigate to the wumpus. In contrast, the fighter in Team C has to randomly choose a direction for shooting when it senses a stench, and cannot get any help from the carrier to locate the wumpus. Consequently, Team C consumes the same number of arrows to get half as many wumpuses as both Team A and Team B, and thus has worse resource utilization.

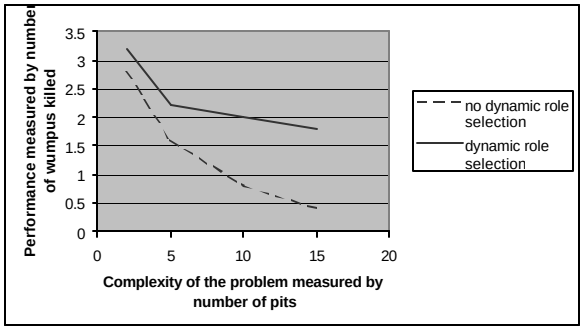

Figure 1 - The Comparison of teamwork with and without dynamic role selection in increasingly complex environments

Experiment 2 was performed on 4 sets of 5 randomly generated maps for a world with 10 by 10 cells, 5 wumpus, and 10 piles of gold. The difference between the sets of maps is that we incrementally increase the complexity of the world by changing the number of pits in the cave, from 2 to 5 to 10 to 15 pits respectively. Each team is allowed to operate for a fixed number of actions. The performance of each team for each case is measured by the same metrics we used in the first experiment. The performance difference is shown in Figure 1.

From Figure 1, we can see that with dynamic role selection, Team E performed better than Team D and the difference becomes larger when the complexity of the

problem scales up. Because the most appropriate fighter will always be chosen to kill the wumpus whenever the carrier finds one, more wumpuses can be killed in a limited number of actions. And in worlds with more pits, it is even harder for the fighter to navigate to the wumpus found, so choosing the closest fighter becomes much more important. But when the world has so many pits that even the carrier gets trapped, teamwork becomes impossible and the performance of both teams will drop significantly.

7 Conclusion

Developing a computational framework for capturing the shared mental model among members of effective teams is a challenging and critical issue for applications ranging from team training to supporting teamwork. The CAST architecture supports flexibility in its teamwork knowledge specification and in dynamic role selection at run time. At the same time, it leverages shared knowledge about the structure and the process of a team to reason about information needs of teammates efficiently. We believe the CAST architecture achieves a reasonable tradeoff between the flexibility and the efficiency for simulating proactive information exchange among teammates. While our experimental results demonstrate anticipated benefits of CAST, it also reveals some limitations of the current CAST implementation. For example, we plan to extend the role selection method for finding backups when an agent dies or becomes non-functional. With such an extension, we hope to simulate more complex teamwork behavior.

8 Acknowledgements

This research described in this paper is supported by a DOD MURI grant F49620-00-1-0326 administered through AFOSR and partially supported by internal seed funds from the College of Engineering through the Training Systems Science and Technology Initiative.

References

[Cannon-Bowers and Salas, 1997] Canon-Bowers, J. A. and Salas, E. A framework for developing team performance measures in training, in Brannick, M. T., Salas, E. and Prince, C.,editors, *Team Performance Assessment and Measurement: Theory, Methods, and Applications*, Lawrence Erlbaum Associates: Hillsdale, NJ. 1997.

[Cohen and Levesque, 1990] Cohen, P.R. and Levesque, H.J. Intention is choice with commitment. *Artificial Intelligence*, 42(3), 1990.

[Cohen and Levesque, 1991] Cohen, P.R. and Levesque, H.J. Teamwork. *Nous*, 25(4):487-512, 1991.

[Grosz and Kraus, 1996] Grosz, B., and Kraus, S. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269-357, 1996.

[Halpern and Moses, 1992] Halpern, J.Y. and Moses, Y. A guide to completeness and complexity for modal logics of knowledge and belief, *Artificial Intelligence*, 54:319-379, 1992.

[Harel, 1984] Harel, D. Dynamic logic. In *Handbook of Philosophical Logic*. Gabbay, D. & Guenther ,F. (eds.). Reidel Publishing : Dordrecht, Netherlands, 1984.

[Ilgen et al., 1993] Ilgen, D. R., Major, D. A., and Hollenbeck J. R. *Leadership and research: Perspectives and Directions*, San Diego: Academic Press, 1993.

[Jennings, 1995] Jennings, N. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. Artificial Intelligence, 75(2):195-240, 1995.

[Jones et al, 1999] Jones, R., Laird, J., Nielson, P., Coulter, K., Kenny, P., and Koss, F. Automated intelligent pilots for combat flight simulation. *AI Magazine*, 20(1):27-41, 1999.

[Rao and Georgeff, 1991] Rao, A.S. and Georgeff, M.P. Modeling rational agents within a BDI Architecture. *Principles of Knowledge Representation and Reasoning, Proceedings of the Second International Conference*, 473-484, 1991.

[Russell and Norvig, 1995] Russell, S. and Norvig P. Artificial Intelligence: *A Modern Approach. Prentice Hall*, New Jersey, 1995.

[Singhal and Zyda 1999] Singhal, S. and Zyda M. *Networked Virtual Environments*. New York, New York, ACM Press, 1999.

[Sowa, 2000] Sowa, J.F. *Knowledge Representation: Logical, Philosophical, and Computational Foundation*. Brooks/Cole: Pacific Grove, CA, 2000.

[Stone and Veloso, 1999] Stone, P., and Veloso, M. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110:241-273, 1999.

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

[Tidhar et al., 1998] Tidhar, G. and Heinze, C. and Selvestrel, M. Flying together: Modelling air mission teams. Journal of Applied Intelligence, 1(1) pp1-1, 1998.

[Wooldridge and Jennings, 1995] Wooldridge, M., and Jennings, N., Intelligence agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115-152, 1995.

[Yin *et al*., 2000]Yin, J., Miller, M., Ioerger, T.R., Yen, J., and Volz, R.A. A knowledge-based approach for designing intelligent team training systems. In: *Proc. of the Fourth International Conference on Autonomous Agents*, pp. 427-434, 2000.