Toward Scalable and Proactive Multi-Agent Teams

John Yen, Jianwen Yin, Michael S. Miller, Dianxiang Xu, and Richard A. Volz
Department of Computer Science
Texas A&M University
College Station, 77843-3112

Abstract

Teamwork becomes very important in many dynamic, multi-agent environments. A multi-agent team can scale up in at least two ways: the size of the team and the complexity of the problem. In this paper, we tackle these issues through proactive information exchange to improve team effectiveness and communication efficiency, and through dynamic task assignment to improve the flexibility of team formation. We also propose a three-tier communication infrastructure to support efficient and flexible communication among team members. We have done some experiments to demonstrate the advantage of proactive information and dynamic task assignment over teamwork.

1. Introduction

Teamwork and team training become very important in many dynamic, multi-agent environments, such as fire fighting team, Command and Control AWACS team, battlefield team, Space Fortress team, etc.[8, 9, 10,12] These teams may have larger team size as well as high complexity of the problem. How to design a scalable multi-agent system to simulate such kind of teams is a very challenging issue. In the CAST (Collaborative Agents for Simulating Teamwork) architecture [11], we address this issue through proactive information exchange and dynamic task assignment. By proactive information exchange, agents can anticipate the information needs of others and provide timely information, and agents can know whom to ask for information when they need any information. This helps make communication more efficient and teamwork more effective. By dynamic task assignment, tasks can be re-allocated among agents and responsibilities can be distributed among agent on-the-fly. So the most appropriate agent can be found for fulfilling certain task, which makes the teamwork more effective especially in a team with large number of agents and a complex problem domain. This also helps balance the workload among team members.

To support proactive information exchange and dynamic task assignment, a basic issue is that we should have a flexible communication infrastructure. Many different communication methods have been developed for different purposes, such as: conflict-resolution [8], coordination [2], distributed plan generation [3], coordination (of individual steps; i.e. synchronization), and maintenance of mutual beliefs about current and achieved goals and commitments [1,10]. There are several different ways for decision on agent communication. Tambe [10] defines a utility for deciding when to communicate. But how to define a utility function in certain domain itself is a hard problem. Based on a three-tier communication mechanism, our approach would be for the agents to simply reason about the team plan itself. We not only allow complicated communication task to be encoded in teamwork language, but also reason about information need of team members in order to support proactive information exchange and dynamic task assignment.

The rest of this paper is organized as follows. Section 2 presents an overview of our approach. In section 3, we elaborate on the three-tier communication mechanism, which is the basis for proactive information exchange and dynamic task assignment. In section 4, we will show some experiment results. Finally we will discuss the contribution of our work as well as some limitations.

2. Overview of Our Approach

2.1 CAST Architecture

CAST (Collaborative Agents for Simulating Teamwork) is a multi-agent architecture that simulates and supports teamwork involving both human and agents. CAST has two goals to achieve. First, it aims to model effective teamwork by capturing *team structures and teamwork processes*. A well-defined team structure enables the team to have predefined roles. A well-defined teamwork process specifies responsibilities of various roles, as well as the goals, strategies, and plans for accomplishing the team's goals. These 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 a team member to reason about the states and the need of his/her teammates.

While both goals are desirable, they conflict with each other. The emphasis on predefined team structure and processes often reduce the flexibility of the team, while maximizing the flexibility of the team usually reduces the

amount of predefined responsibilities for members of the team. To balance these two conflicting goals, we focus on two specific effects 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.

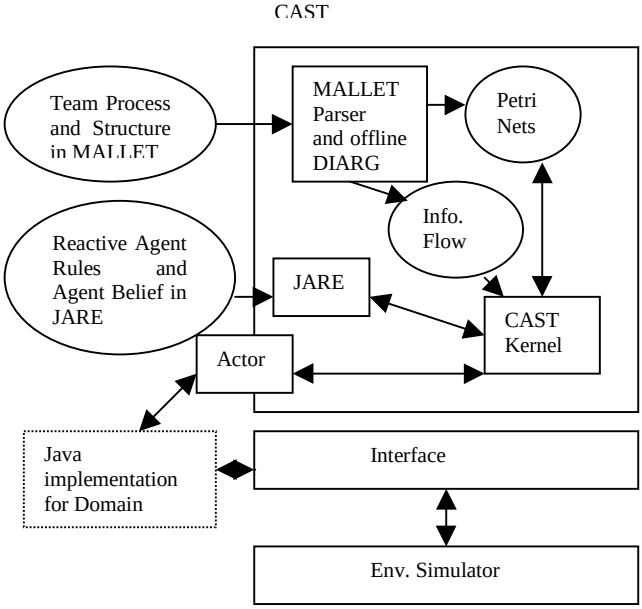

Figure 1. CAST Architecture

The Petri Net is a natural representation for parallel action and synchronization in a multi-agent world [6]. 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*. Control nodes represent the belief an agent has about the current actions being performed by others in the team. We note that belief nodes can represent first-order knowledge, which is adequate for many training applications. From the viewpoint of BDI (belief, desire, intentions) agents, markings and places in Petri Net are efficient representation of an agent's belief about its teammates. In addition to serving as the shared mental model, Petri Nets in CAST also play the dual role of monitoring and tracking the execution of team plans.

As shown in Figure 1, CAST generates Petri Net-based representation according to team specification in MALLET (a Multi-Agent Logic-based Language for Encoding Teamwork), which contains 1) team structures (roles and responsibilities), and 2) teamwork process knowledge (e.g., individual plans, team plans). The CAST kernel enables the CAST agents to decide on the fly how to accomplish desired goals, how to select responsibilities to commit or delegate, how to proactively assist others in the team, and how to effectively communicate with the team.

2.2 Proactive Information Exchange

Different communication protocols can be selected for information exchange for different kinds of information. Intuitively in a team, when any team member needs any information, they just ask for it. That is the simplest way for information exchange. They just need to figure out whom to ask and when to ask. This may reduce the communication overhead in the sense of only needed information is exchanged. But there are two major disadvantages for this way of information exchange: 1) Every time when agents need certain critical information, they need to ask and wait for an answer, which may waste time or even fail in time critical situation. 2) Whenever there are multiple candidates who can answer the question, agent who needs the information may pick the wrong teammate to ask for the question. This may cause failure either because the agent being asked is too busy and delayed the answer too long or because the agent being asked does not have the updated information in that situation. So we believe in a better team, agents should be able to anticipate the information needs of others and proactively provide information to others who need the information. This certainly solve the two issues of applying only *ask*. People may argue that this may cause other problems such as: 1) Agents may keep sending similar information whenever there is a change; 2) Different agents may pass the same information around. This is true and this is where we make our point: we need to use different communication protocols for different kinds of information.

We differentiate information into the following two types:

- Infrequently changing information, but frequently needed information, let us call it I1;
- Frequently changing information, but infrequently needed information, let us call it I2.

We start by defining a simple belief language and model theory to be able to talk about the mental states. We model beliefs using a modal operator BEL, e.g. `(BEL Ca (have Fi arrow))`, with the usual possible worlds semantics [1]. 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 (i.e. operators to be executed), to which agents can make commitments. Similarly we define another modal operator KNOW, e.g. (KNOW A x) meaning that A knows about x.

Using this framework, we can formally characterize the (normative) conditions under which information exchange should take place. If information I falls into the category of I1, then information *I1* should be sent from one agent *A* to

another agent *B* when: 1) agent *A* knows the truth-value of *I1*, 2) agent *A* believes that agent *B* does not currently know *I1*, and 3) *B* has a current goal *G* that depends on knowing *I1*, i.e. if *B* does not believe *I1*, then it will never be able to accomplish it's goal, but if it knew *I1*, it would be able to:

$$(\text{BEL A I1}) \wedge (\text{BEL A } \neg(\text{BEL B I1})) \wedge$$
$$(\text{BEL A (GOAL B G)}) \wedge$$
$$[\neg(\text{BEL B I1}) \rightarrow \Box\neg(\text{DONE B G})] \wedge$$
$$[(\text{BEL B I1}) \rightarrow \neg\Box\neg(\text{DONE B G})]$$
$$\rightarrow (\text{GOAL A (Inform B I1)})$$

where □ is the temporal operator for 'always'. In CAST, we use the pre-conditions of operators to determine the information *I1* that an agent needs to know to achieve its goals. Note, the communication is suppressed only when (*A* believes that) *B* already believes *I1* is true; but if *B* does not have a belief about the truth value of *I1* 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. As in real teams, efficiency may go down if the team members do not have the same overall mental model of the team.

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 on their own 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 (inference in almost any non-trivial logic generally has high computational complexity), 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.

If information I falls into the category of I2, then agent A should ask agent B about information I2 when: 1) agent *A* does not currently know the truth-value of *I2*, 2) agent *A* believes that agent *B* currently know *I2*, and 3) *A* has a current goal *G* that depends on knowing *I2*, i.e. if *A* does not believe *I2*, then it will never be able to accomplish it's goal, but if it knew *I2*, it would be able to:

$$\neg (\text{BEL A I2}) \wedge (\text{BEL A (KNOW B I2)}) \wedge$$
$$(\text{BEL A (GOAL A G)}) \wedge$$
$$[\neg(\text{KNOW A I2}) \rightarrow \Box\neg(\text{DONE A G})] \wedge$$
$$[(\text{KNOW A I2}) \rightarrow \neg\Box\neg(\text{DONE A G})]$$
$$\rightarrow (\text{GOAL A (activeAsk B I2)})$$

And B should reply to A upon receiving the activeAsk request if B really knows it, we will introduce the DIARG algorithm [11,12].

2.3 Shared Mental States

In order for agents in a team to exchange information proactively, to coordinate and cooperate, agents should share certain information. Joint intention [1] and shared plans [7] can all be regarded as part of the shared mental models. In our approach, we assume our agents have the following information in their shared mental model: static information, such as knowledge of team structure, knowledge of responsibilities, knowledge of capabilities, knowledge of team goals, knowledge of team plans, and knowledge of information flow. And dynamic information, such as knowledge of the state of each agent, including which plan an agent is executing, where agents could possibly be in their team plans, etc.

The static information of team structure, responsibilities, capabilities, team goals all come directly from the MALLET team specification. The MALLET parser reads in the MALLET file and generates Petri Net from its goal related plans. Information flow is represented through a 3-turple: <Info, Providers, Needers>. Information flow of type I1 and I2 information can be detected through offline DIARG algorithm and stored in different data structure. The dynamic information about the status of each agent in its plan is tracked through Petri Net. The Petri Net is not only used as the shared mental model for the static information of team plan, but a tool to keep track of the plan execution. The static knowledge of the process of team plan is maintained in the Petri Net hierarchically and the detailed information is also represented in the Petri Net.

Each goal related team plan in MALLET becomes a Petri Net. This Petri Net has a start node and consists of transitions that are either actions or conditional tests by the agent that is executing that Petri Net. A Petri Net may not always be executed by a single agent, sometimes different parts of the Petri Net can be executed by different agents (e.g. parallel execution or sequential execution). Therefore once an agent has completed its part of the plan as embodied in the Petri Net, it must hand over control if there follows a portion that is to be executed by another agent. So by having the shared knowledge of the team plan as represented in MALLET, the first agent knows whom to tell whenever it finishes its own part of the process in the current plan, and the second agent knows from whom to expect the information of process control.

2.4 Scalability via Dynamic Task Assignment

A role serves as a preference constraint in assigning agents to responsibilities. Associating a role to a generic agent (i.e., an agent variable) with certain responsibilities is called a role specification, while associating a role to a actual agent is called a role selection or task assignment. There are two kinds of role specifications: (1) global role specification, and (2) plan-specific role specification. A global role specification describes a role the generic agent can play in all situations (called responsibility in

MALLET). A plan-specific role specification serves as a constraint for selecting agents in a plan.

One of the most important issues in modeling team-based agents is selecting agents for roles (i.e., for actions or plans). Roles in CAST are specified with some constraints about the agents who can take the responsibility. For example, a specific participant of a team plan may be constrained to agents with certain attributes. If a team consists of one carrier (who scouts) and ten fighters (who shoot), an instantiation of the team uniquely determines the agent who will take the responsibility of scout in the plan, but leaves the choice for fighter unclear. This leaves an ambiguity about which fighter should actually shoot the wumpus in a particular wumpus hunt. This ambiguity may be resolved through a coordination protocol (e.g., a fighter who first volunteers gets to shoot the wumpus) or through a commonly understood criteria (e.g., a fighter who is closest to the location of the wumpus as illustrated in Section 4). Obviously, the second criteria can work only if all fighters have the same assessment about each fighter's distance to the wumpus. Like role specification, a role assignment can be made globally or locally in a plan. A plan-specific role specification allows different instantiations of a plan to have different agent assigned to a particular role in the plan. For instance, the fighter for a wumpus found can be different from that of another wumpus. This is a situation of "multiple legal candidates" for a responsibility. This is similar to XOR responsibility mentioned in our previous paper.

In general, role selection can be made in several ways: (1) choose the agent (e.g., randomly or intelligently) at the compile time, (2) choose the protocol for determining the assignment at compile time, select the agent based on the protocol at run time, (3) choose the protocol as well as the agent at run time. The current CAST implementation provides a volunteer protocol for role selection of unconstrained roles and a dynamic role selection protocol for role selection of constrained roles; but other protocols can also be added. A problem related to dynamic role selection is that of team reconfiguration due to losing a portion of the team. Under such circumstances, a back up agent needs to be assigned to the responsibility.

In order to introduce the dynamic role selection algorithm, we need to first introduce several concepts about plans and operators in MALLET. In MALLET, there are two types of plans: i*ndividual* plans (*I-plan*) and *team* plans (T-plan). And there are two types of operators: *individual* operators (*I-op)* and *team* operators (*T-op*). Individual plans and individual operators are assumed to be executed by only one agent at a time. However, team plans and 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

The basic idea of the current dynamic role selection protocol is: whenever an agent start a team plan, for all roles not constrained by any constrained condition, agent will use the volunteer protocol to select one agent to fit in that role; for all roles constrained by any constrained condition, postpone the role selection until starting an individual plan or individual operator involving that role. Whenever an agent encountering a step in its plan body (the step could be another team plan, a team operator, an individual plan or an individual operator), the agent will use the following algorithm to decide who is going to take the role:

role-selection(step A)

1) *If A is a Team Plan return;*
2) *If all roles involving in A has been bound to certain agents in the plan A comes from then return;*
3) *If self can not play any role which has not been bound to any agent in the plan A comes from, then return;*
4) *For all other cases, call select-role(step A) to decide who will take the untaken roles*

select-role(step A)

1) *For any role that has not been bound to any agents in the plan A comes from*
2) *Let CANDIDATE be all the agents that can play that role;*
3) *Remove from CANDIDATE any agents that is busy;*
4) *If CANDIDATE is empty, then return;*
5) *Let ROLE-BINDING be a partial role-binding traversed from the plan A comes from and include any commitment made so far;*
6) *Let CONSTRAINT be the constrain condition on the role;*
7) *Let CANDIDATE = checkConstraint (CANDIDATE, ROLE BINDING,CONSTRAINT), the checkConstraint is a domain related method which return a new list of candidate;*
8) *If CANDIDATE is empty then return;*
9) *If A is an I-plan, I-op, XOR-op or OR-op then*
   let agent x be the first one in CANDIDATE;
   set commitment(A) = x;
   set x as busy;
   return;
10) *If A is an AND-OP then*
   set commitment(A) = all agent in CANDIDATE;
   set all agent in CANDIDATE as busy;
11) *If ROLE-BINDING is a complete binding, and the agent plays one of the roles, then agent takes the step A.*

The role binding information should be traversed vertically but not horizontally, which means that the role binding information and dynamic role selection information will be kept within the scope of current sub-plan, if agent move to the next sub-plan, it needs to do the dynamic role selection again. For example, in the following team plan:

```
(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))))
)
```

whenever a fighter is selected for ?fi in (do ?fi (moveto-wumpus)), the role binding will be kept within the individual plan without further role selection, but after the step is finished, the dynamic role selection algorithm should be called again for (do ?fi (kill-wumpus)), which is unnecessary in this particular example.

The major benefits of dynamic roles selection we have demonstrated are better performance especially in high complex environment, relatively balanced workload, and thus better scalability. Certainly, in order to do dynamic role selection, agents need some extra communication.

3. Communication Support for Agent Teams

3.1 Types of communication

Communication in our approach serves four purposes for effective teamwork:

- Proactive information providing;
- Information request and reply;
- Disambiguating a shared responsibility;
- Control token propagating.

The first type of communication is very important for agent to anticipate the information needs of other agents and proactively provide information to others. By having a shared mental model of teamwork, agent can keep track of the teamwork process as well as the current states, agent can figure out what kind of information others may need.

Information request and reply is mainly applied for the type I2 information. Whenever agent needs certain information to fulfill its goal, it can apply DIARG algorithm to figure out who can provide the relevant information and use this kind of communication to get the information. There is always trade-off about which method to apply depending on the differentiation of type I1 and type I2. This information should be provided by domain experts.

When there are multiple candidates for one task (for example, the role played by multiple agents involved in a plan), then there is ambiguity about who is going to really take action toward the plan. In this case, our CAST agent can use volunteer for roles not constrained by any conditions and dynamic role selection algorithm for role with constraint.

Agents in a team need to be aware of each other's steps, such as the control sequence. In real world, agents may know part of it by observation, and part of it through communication. Since we use Petri Nets to model the execution of plans, we should provide communication for propagating control tokens among agents.

Moreover, there are other communication needs in teamwork, such as synchronization, which we will not elaborate in this paper.

3.2 Communication protocols

Our three-tier communication mechanism consists of:

1) Low level communication methods based on Java RMI;
   - Broadcast and wait
   - Broadcast and do not wait
   - Multicast and wait
   - Multicast and do not wiat
   - Unicast (one-to-one) and wait
   - Unicast and do not wait
2) Atomic level communication operators;
   - Tell (multicast and do not wait)
   - Ask (unicast and wait)
   - Reply (unicast and do not wait)
   - Selec-role (broadcast and do not wait)
   - Control-token (broadcast and do not wait)
3) High level communication protocols.
   - ProactiveTell through Tell
   - ActiveAsk through Ask and Reply pair
   - Volunteer through Select-role
   - Dynamic role selection through Ask, Reply and Select-role (after calculation)
   - Coordinating information propagation through Control –token mechanism

There are three most popular communication methods: broadcast, multicast and unicast. Broadcast allows agents to send the information to all team members within a team, including itself. It is useful for team coordination and control the sequential actions among team members. Multicast allows agents to send information to a partial team. It can make teamwork more efficient when only partial team is involved in a task. The information is only important to the partial team. Unicast allows communication happens between two parties without disturbing anyone else. For each of the method, there are two situations : wait for a reply or acknowledge, or do not wait for anything. For important information or when agent wants to ask for certain information, agent needs to know whether information is successfully sent in the first situation, or agent needs to get reply in order to go on. In both case, agent need to wait. But there are other cases that agent just provide certain information and go on with its own task without waiting for any response.

For the second tier, it is similar to those KQML performatives. Tell multicast information to a sub-group of agents without waiting for a response. Ask and Reply are used together for agents to query and pass information between each other in a team. Select-role is used to announce the role selection of an agent within the whole team. And Control-token is used to announce the status of agents within a team plan.

The third tier is the combination of several atomic communication operators with certain algorithm to serve a more complex purpose. ProactiveTell is used to anticipate information needs (through DIARG) of other and Tell them the information. ActiveAsk is to figure out its own information needs and who can provide the information (through DIARG) and ask for the information and wait for an Reply. Volunteer is for agents to volunteer for a certain role which is not constrained by any condition and inform others to disambiguate among agents about shared responsibilities. And dynamic role selection is used for agent to announce its selection on certain role constrained by condition for the same purpose of disambiguating. But instead, agents needs to get certain information to do so. So it needs to go through Ask-wait for Reply- calcuation-Select-role. Control token propagating is used to propagate control tokens among agents so that agents can keep track the states of other team members and the whole team.

4. Experiments

Candidate team domains could be military teams, fire fighting teams, RoboCup soccer teams, space control teams, etc. The domain itself is not very important as long as it meets the following criteria. The team should be well structured, the role of each team member should be well defined, the capabilities of each team member should be well defined, the responsibilities of each team member should be well defined, there should be information needs among agents, team plans should be pre-defined, and communication is assumed to have cost.

For simplicity, we decided to choose multi-agent Wumpus World as our team domain, which meets our criteria and has a very easy to understand domain knowledge. And we will find out that it is also very convenient for the research purpose. We have constructed a testbed application based on the multi-agent Wumpus World, in which we implement two roles: carrier and fighter.

Agents can communicate in three ways: (1) proactive tell, (2) broadcast information, and (3) broadcast control tokens for coordination purpose.

We can evaluate the following four aspects of the team:

- Team performance: (1) the number of wumpus killed, (2) the number of arrows used, (3) the amount of gold gathered, (4) the number of actions taken, (5) the number of agents killed.
- Communication overhead: (1) the number of broadcast messages, and (7) the number of proactive tell messages.

In order to evaluate the effectiveness of architectural supports from CAST, we have designed two sets of experiments. In the first set of experiment, in order to show the advantage of proactive information exchange, we have devised three different multi-agent teams for comparison. Each team consists of a fighter and a carrier. Team A and Team B uses a MALLET specification of a team plan that requires the climber to find a wumpus, then navigate a fighter to a cave adjacent to the wumpus and shoot the wumpus. The three teams use the same knowledge base (i.e., JARE rules) for reasoning about the environment and for determining the priority of actions. Their differences are listed below:

1) Team A: CAST agents with both shared mental model and DIARG support. Proactive information exchange is being done.
2) Team B: CAST agents with shared mental model, but not DIARG support. Instead, each agent broadcasts every new piece of information found.
3) Team C: Agents without shared mental model. They are simply a group of independent agents.

The experiments were performed on five randomly generated maps for a world with 10 by 10 cells, 5 wumpus, 2 pits, and 10 piles of gold. Each team is allowed to operate a fix amount of time. The performance of each team for each test case is measured by items listed in Table1. The results of the experiments are summarized in Table 1.

Table 1. Experiment Results

Team	V1	V2	V3	V4	V5	V6	V7
Team A	W1	4	4	142	0	4	5
	W2	5	5	138	0	4	5
	W3	4	4	145	0	4	4
	W4	2	2	152	0	4	3
	W5	4	4	149	0	4	3
Team B	W1	5	9	144	0	226	0
	W2	5	9	149	0	220	0
	W3	4	4	144	0	170	0
	W4	4	4	155	0	196	0
	W5	3	10	80	0	134	0
Team C	W1	5	14	250	0	0	0
	W2	1	2	250	0	0	0
	W3	3	17	250	0	0	0
	W4	4	21	250	0	0	0
	W5	2	14	250	0	0	0

V1: test cases (worlds) V2: # wumpus killed
V3: # arrows used V4: # actions
V5: # dead agent V6: # broadcast messages
V7: # proactive info exchanges

As shown in the Table 1, Team A and Team B achieves comparable performance, because they use an identical shared mental model specified in MALLET. However, Team B has a much higher communication overhead. The main advantage of Team A is that teamwork performance is improved without significantly increasing communication overhead. For example, the fighter only needs one arrow to shoot a wumpus with known location. This is achieved because the carrier agent tells the shooter agent proactively about the location of the wumpus, because DIARG infers that the fighter needs the information to navigate to the wumpus. In contrast, the fighter in Team C has to randomly

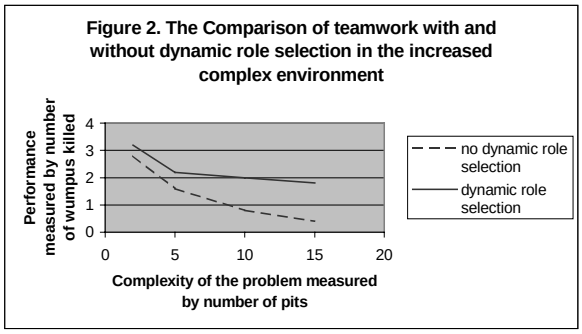

Figure 2. The Comparison of teamwork with and without dynamic role selection in the increased complex environment

choose a direction for shooting when it senses stench. Consequently, Team C consumes more arrows on the average than both Team A and Team B.

In the second set of experiments, to show the benefit of dynamic role selection algorithm, we have devised another three different multi-agent teams for comparison. Each team consists of a carrier and three fighters. They all use a MALLET specification of a team plan that requires the carrier to find a wumpus, then navigate a fighter to a cave adjacent to the wumpus and shoot the wumpus. The three teams use the same knowledge base for reasoning about the environment and for determining the priority of actions. And they all use the same team plan knowledge in MALLET. Their differences are listed below:

1) Team D: CAST agents with both shared mental model and DIARG support. Proactive information exchange is being done. But agents just neglect the constraints on roles and do not do dynamic role selection. An agent team is formed before starting the team plan. (This is equivalent to Team A, but since the team structure is different and we run a new set of test, let's use D for the time being.)
2) Team E: CAST agents with shared mental model and DIARG support, agents consider the constraint condition and dynamically select roles based on the constraint condition on roles. In order to do this, besides proactive information exchange, agents need to use activeAsk to capture the location information in order to check the constraint condition.

The experiments were first performed on five randomly generated maps for a world with 10 by 10 cells, 5 wumpus, 2 pits, and 10 piles of gold. Each team is allowed to operate a fixed amount of time. The performance of each team for each test case is measured by the same characteristics. The results of the experiments are summarized in Table 2. The number of actions each agent performed and total number of actions performed by each team is summarized in Table 3. Though we listed several items in the following table to measure the performance, the most important are the number of wumpuses killed and the number of communication and actions taken. The other measures could be uncertain due to random layout of the cave and the location of agents. For example, carrier can pick up gold when it walks around and bumped into a pile of gold, fighter can shoot the wumpus with one arrow when it knows the location of the wumpus or randomly shoot certain direction when it senses stench without knowing the exact location of wumpus. Hence we will focus our discussion on the important measures.

Table 2. Team Performance

	V1	V2	V3	V4	V5	V6	V7
Team D	W1	2	2	53	0	4	3
	W2	2	2	55	0	4	2
	W3	4	4	49	0	4	3
	W4	4	15	50	0	4	3
	W5	2	4	63	0	4	1
	A	2.8	5.4	54	0	4	2.4
Team E	W1	3	3	45	0	56	9
	W2	4	4	47	0	48	9
	W3	3	3	38	0	76	12
	W4	4	4	53	0	30	9
	W5	2	2	34	0	73	9
	A	3.2	4	43.4	0	57	9.6

Table 3. Action Counts

	V1	CA	Fi1	Fi2	Fi3	Total	W
Team D	W1	46	7	0	0	46	2
	W2	42	13	0	0	55	2
	W3	15	34	0	0	49	4
	W4	41	14	0	0	55	3
	W5	3	60	0	0	63	2
	A	29	25.6	0	0	53.6	2.6
Team E	W1	32	11	2	0	45	3
	W2	29	0	13	5	47	4
	W3	26	0	9	3	38	3
	W4	45	2	1	5	53	3
	W5	25	0	0	9	34	2
	A	31	2.6	5	4.4	43.4	3

As shown in Table 2, Team E benefits from the dynamic role selection algorithm in 1) getting higher performance by choosing the most appropriate agents to do the job; 2) balancing the workload among agents by selecting different agents to do similar job at different time. But to guarantee dynamic role selection, agents need more communication. There is always tradeoff for teamwork. In teamwork which communication cost is high (compared with other actions), we may work without dynamic role selection and only use

proactive information exchange. In a team which communication cost is relatively low, we may consider dynamic role selection to improve the team performance by

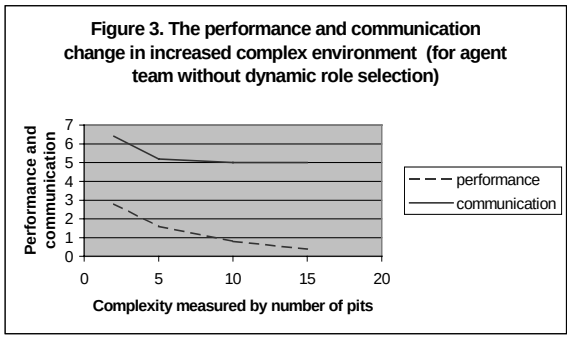

Figure 3. The performance and communication change in increased complex environment (for agent team without dynamic role selection)

reducing the actions taken by agents and balance the workload among agents playing same roles.

More experiments have been done on team D and E to show the performance change with the increasing complexity of the problem. W have the following observation with the complexity increasing:

- The communication load keeps stable;
- The performance dropped significantly for Team D;
- The performance keeps stable for Team E.

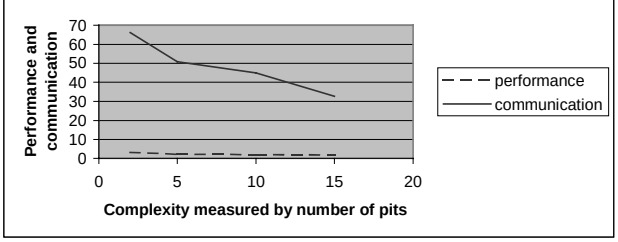

Figure 4. The performance and communication change in an increased complexit environment (for a multi-agent team with dynamic role selection)

Figure 3. shows that the performance decreases with the increasing of the complexity of the problem in Team D, but the communication load does not change much.

Figure 4. shows that the performance keeps stable with the increasing of the complexity of the problem in Team D, and the communication load does not change much. Series 1 stand for performance and Series 2 stand for communication.

When there are 20 pits, the carrier can hardly move around and find any wumpus, so the performance for both cases will drop to near zero, that is a problem even human teams do not have good solutions for.

5. Conclusions

In this paper, we have presented proactive information exchange and dynamic role selection for task assignment. And a three-tier communication infrastructure has been introduced to support these mechanisms. Finally our experiments have demonstrated that with the support of proactive information exchange and dynamic role selection, the teamwork becomes more efficient especially when the problem scales up in complexity. However, we do not deal with plan failure in this paper. We need to further extend our dynamic task assignment to handle the failure conditions. But we believe our approach will be advantageous in reducing communication overhead and allowing flexible team formation building to support effective teamwork.

References

[1] P. R. Cohen, and H. J. Levesque, "Teamwork," *Nous 25(4), Special Issue on Cognitive Science and Artificial Intelligence*, 1991, pp. 487-512.

[2] K. S. Decker, K. S., and V. R. Lesser. "Designing a Family of Coordination Algorithms". In *Proceedings of the First International Conference on Multi-Agent Systems*,73-80, 1995.

[3] E. H. Durfee, and V. Lesser, "Using Partial Global Plans to Coordinate Distributed Problem Solvers", In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 875-883, 1987.

[4] B. Grosz, "Collaborating Systems", AI Magazine, 17(2), 1996.

[5] T. Ioerger, R. Volz, R., and J. Yen, "Modeling Cooperative, Reactive Behaviors on the Battlefield Using Intelligent Agents", In *Proceedings of the Ninth Conference on Computer Generated Forces* (9th CGF), 13-23, 2000.

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

[7] K. Sycara, "Multiagent Compromise Via Negotiation", In Gasser, L. and Huhns, M., editors, *Distributed Artificial Intelligence*. Morgan Kaufmann: Los Altos, CA, 1989.

[8] M. Tambe, "Agent architectures for flexible, practical teamwork," in *Proceedings of the 14th National Conference on Artificial Intelligence*, Providence, Rhode Island 1997, pp. 22-28.

[9] M. Tambe, W. L. Johnson, R. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb, "Intelligent agents for interative simulation environments," *AI Magazine*, 1995.

[10] M. Tambe, "Towards Flexible Teamwork", Journal of Artificial Intelligence Research, 7(1), pp.83-124, 1997.

[11] J. Yen, J. Yin, T. R. Ioerger, M. S. Miller, D. Xu, and R. A. Volz, "CAST: Collaborative Agents for Simulating Teamwork," will appear in Proc. of the International Joint Conference on Artificial Intelligence, *IJCAI-2001*.

[12] J. Yin, M. S. Miller, T. R. Ioerger, J. Yen, and R. A. Volz, "A Knowledge-Based Approach for Designing Intelligent Team Training Systems," in *Proceedings of the Fourth International Conference on Autonomous Agents*, pp.427-434, 2000.