Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 63–77. http://www.scpe.org
ISSN 1895-1767

OBSERVATION-BASED PROACTIVE COMMUNICATION IN MULTI-AGENT TEAMWORK

YU ZHANG*

Abstract. Multi-agent teamwork is governed by the same principles that underlie human cooperation. This paper describes how to give agents the same cooperative capabilities, observability and proactivity, that humans use. We show how agents can use observation of the environment and of teammates' actions to estimate the teammates' beliefs without generating unnecessary messages; we also show how agents can anticipate information needs among the team members and proactively communicate the information, reducing the total volume of communication. Finally, we present several experiments that validate the system developed, explore the effectiveness of different aspects of observability and introduce the scalability of the use of observability with respect to the number of agents in a system.

Key words. Multi-agent systems, teamwork, agent communication, observability

1. Introduction. Recently, the focus of much research on multi-agent systems (MAS) has shifted from strong agency [26] to teamwork, which is a cooperative effort by a team of agents to achieve a common or shared goal [23]. Research on multi-agent teamwork builds on findings about effective human team behaviors and incorporates them into intelligent agent technologies. For example, the shared mental model, one of the major aspects of the psychological underpinnings of teamwork, has been adopted widely as a conceptual basis of multi-agent teamwork. Based on the shared mental model, an effective team often can anticipate the information needs of teammates and offer pertinent information proactively [18, 22]. Consequently, supporting proactive information exchange among agents in a multi-agent teamwork setting is crucial [29]. Substantial challenges arise in a dynamic environment because agents need to deal with changes. Although partial observability of dynamic, multi-agent environments has gained much attention [17, 11], little work has been done to address how to process what is observable and under which conditions; how an agent's observability affects the individual's mental state and whole team performance; and how agents can communicate proactively with each other in a partially observable environment.

In this paper, we focus on how to include represent observability in the description of a plan, and how to include it into the basic reasoning for proactive communication. We define several different aspects of observability (e.g., seeing a property, seeing another agent perform an action, and believing another can see a property or action are all different), and propose an approach to the explicit treatment of an agent's observability that aims to achieve more effective information exchange among agents. We employ the agent's observability as the major means for individual agents to reason about the environment and other team members. We deal with communication with the 'right' agent about the 'right' thing at the 'proper' time in the following ways:

- Reasoning about what information each agent on a team will produce, and thus, what information each agent can offer others. This is achieved through: 1) analysis of the effects of individual actions in the specified team plans; 2) analysis of observability specification, indicating what and under which conditions each agent can perceive about the environment as well as the other agents.
- Reasoning about what information each agent will need in the process of plan execution. This is done through the analysis of the preconditions of the individual actions involved in the team plans.
- Reasoning about whether an agent needs to act proactively when producing some information. The decision is made in terms of: 1) whether or not the information is mutable according to information classification; 2) which agent(s) needs this information; and 3) whether or not an agent who needs this information is able to obtain the information independently according to the observation of environment and other agents' behaviors.

We also present several experiments that validate the system developed, explore the effectiveness of different aspects of observability and introduce the scalability of the use of observability with respect to the number of agents in a system.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 is an overview of the system architecture, which is called CAST-O. Section 4 discusses how an agent's observability is represented, and how an agent's beliefs are maintained in the course of observations. Section 5 describes observation-based

*Department of Computer Science, Trinity University, San Antonio, TX, 78216, USA.

proactive communication among agents. Section 6 is an empirical study based on a multi-agent Wumpus World. Section 7 summarizes our work and discusses issues for further research.

2. Related Work. A single agent's observability and reasoning have received researchers' attentions for some time. Perception reasoning is one of these research directions [16, 24]. For example, "seeing is believing" has been adopted for perception-based belief reason[2, 13]. In recent years, observability has been used widely to understand behaviors of multi-agent systems. One study of particular interest is a logic for visibility, seeing and knowledge (VSK), which explores relationships between what is true, visible, perceived, and known; it also investigates a number of interaction axioms among agents, such as under which condition agent a sees everything agent b sees or agent b knows everything agent a sees [27]. However, VSK logic does not address two major issues regarding agent cooperation: 1) an agent uses the effects of actions in reasoning what others are likely to know, but VSK does not provide a way to treat actions through observation; 2) VSK does not provide agents with an effective way to utilize their observation to manage communication. Isozaki and Katsuno propose an algorithm to reason about agents' nested beliefs (which are one's belief about the belief of another), based on observatio[10]. However, they do not represent the process of observation, such as what can be seen and under which conditions. Tambe and Kaminka use observation to monitor failed social relationships between agents [12], but they do not give details about how agents' belief about their teammates' mental states are updated. Viroli and Omicini devise a formal framework for observation that abstracts conditions that cause agents' interactive behavi [25]. But, they don't say much about how the observation to environment is processed. All of above fall into the category "passive observation", in the sense that each agent evaluates observability conditions at the appropriate times. Our work also belongs to passive observation. However, we aim to reduce the amount of communication by reasoning about agent observability, the capability to observe environment and actions. We relate an agent's observability to its mental state, and then use observation and belief about others' observabilities to estimate its teammates' mental states. That is, an agent can exploit knowledge about what it and its teammates can see to help decide when others might or might not know some information. Ioerger has considered "active observation", in which he invokes additional 'find-out' plans to seek values for unknown conditions knowledge of whose values would enable situation assessment [9].

To date, control paradigms for cooperative teamwork have allowed agents to communicate about their intentions, plans, and the relationships between them [23, 21]. However, this complex team cooperation behavior requires high-frequency communication and computation time, which weakens teamwork efficiency. Moreover, some researchers have found that communication, while a useful paradigm, is expensive relative to local computation [1]; therefore techniques that reduce extraneous communication during teamwork processes are of particular importance. On the other hand, there exist several communication-less agent cooperation techniques such as social conventions [20], focal points [14], plan recognition [8], decision-theoretic modeling [15, 28], and game-theoretic recursive modeling [5]. In general, these techniques emphasize inferring others' actions implicitly or explicitly, based on established norms for behavior or on knowledge about the preferences or interests of others. However, strategies such as social conventions or focal points totally eliminate communication and use convention rules to guide agents' actions, strategies such as plan recognition or decision-theoretic normally have high computational complexity in dealing with uncertainty which weakens teamwork efficiency, and game-theoretic recursive modeling is primarily suitable for two-member teams. Our approach to proactive communication is different in that agents are capable of predicting team-related information (by analyzing team plans) and distributing such information only when it is necessary. The communication need is reduced, by using belief of what agents can observe, and hence don't have to be told.

3. The CAST-O Architecture. The CAST-O architecture is an extension of CAST (Collaborative Agents for Simulating Teamwork) [29]. There are three aspects to the extension: 1) representation of agent observability about the environments and other agents' actions; 2) belief-maintenance in terms of observation; 3) observation-based proactive communication among agents.

An agent team is composed of a set of agents. The team members share the team knowledge that is represented in MALLET (Multi-Agent Logic Language for Encoding Teamwork), which provides descriptors for encoding knowledge about teamwork processes (i. e. individual/team plans and operations), as well as specifications of team structures (e.g., team members and roles) [30]. Each agent has an individual knowledge base (KB) to specify its beliefs about the environment and beliefs about teammates' mental states. The environment simulation provides an interface through which the agents can interact with the environment. In

the process of plan execution, individual agents can observe the environment and their teammates' behaviors, infer the teammates' mental states, communicate with each other, and perform actions.

Plans are at the center of activity. They describe how individuals or teams can go about achieving various goals. Plans are classified into individual plans and team plans. Each individual plan has a process consisting of a set of operations, each of which is either a primitive operator, or a composite operation (e.g., a sub-plan). Team plans are similar to individual plans, but they allow multiple agents or agent variables to be assigned to carry out operations or plans (some of the requiring a team). A DO statement is used to assign one or several agents to carry out specific operators or sub-plans. The following is an example team plan for the multi-agent version of Wumpus World (refer to section 6 for more details):

```
(tplan killwumpus()
  (process
    (par
      (seq
        (agent-bind ?ca (constraint (play-role ?ca carrier)))
        (DO ?ca (findwumpus ?w))) // carrier is assigned
      (seq
        (agent-bind ?fi (constraint ((play-role ?fi fighter)
                        (closest-to-wumpus ?fi ?w))))
        (DO ?fi (movetowumpus ?w)) // fighter who is closest to
                                   // wumpus is assigned
        (DO ?fi (shootwumpus ?w)))))) // shootwumpus is an operator
```

where `findwumpus` and `movewumpus` are individual plans, and `shootwumpus` is an individual operator specified as follows:

Generally, operators are defined by their preconditions and effects, which are logical conjunctions. An individual action is the execution of an instantiated operator in a DO statement. It is represented as:

```
<action> ::= (DO <doer> (<operator-name> <args>))
```

where `<doer>` is the agent assigned to the action and `<operator-name>` and `<args>` are correspondent to the name and arguments of the operator. Sample individual actions in the extended Wumpus World are as follows:

```
(DO ?fi (shootwumpus ?w))
(DO ?ca (pickupgold ?g))
```

We assume that the precondition of the action must be believed by <doer> before the action can be performed and the effect must be believed after the action is performed. Since actions are domain-dependent, when agents perform the actions, they send a signal to the environment simulation. Then the actions are visible to any team member whose observability (see section 4) permits it at the time the actions are performed.

An essential feature that differentiates an agent team from a set of individual agents is that a team of agents may perform a joint action, which is the union of simultaneous individual actions performed by individuals sharing certain specific mental properties [4]. MALLET provides a descriptor joint-do for agents performing the joint action, and specifies three different joint types: AND, OR or XOR [29]. For example, we may define following joint action in the extended Wumpus World:

```
(joint-do AND
   (DO ?ca (move ?x ?y))
   (DO ?fi (move ?x ?y)))
```

which means agents `?ca` and `?fi` move simultaneously.

Given a team plan expressed in MALLET, we can explicitly deduce information needs and production from the pre-conds and effects of operators and implicitly deduce others from the plan structure, e.g., joint-do requires coordination regarding starting time, or operations in parallel need coordination in terms of the starting and ending of the par set of branches. The latter, for example, might be determinable from observations, avoiding the need for explicit communication. In addition, if multiple agents are capable of performing the same tasks, the MALLET team plan is likely to contain agent selection criteria (e.g., the closest agent to a wumpus should kill it). Again, this falls in the realm of implicitly determinable coordination communication. While this paper has focused on the only the explicitly determinable part of this (i. e., things derived from pre-conds and effects conditions), the basic structure of the use of observation can be applied to more general situations.

Another important setting for agents' teamwork is environment. The environment is composed of objects. Each object has some properties. A property is represented as follows:

```
<property> ::= (<property-name> <object> <args>)
<object>     ::= <agent>|<non-agent>
```

where <object> could be either agent or non-agent, and <args> is a list of arguments describing the property. Sample properties in the extended Wumpus World are as follows:

```
(location fi ?x ?y),
(dead w1 ?state).
```

The usefulness of properties derives from treating them as queries to the environment, using variables for any or all of the arguments. Unification will provide values, if any, for the free variables that make the query true; if there are no such values, then the value for the query will be false.

During a teamwork process, the environment simulation provides an interface through which the agents can observe the environment and their teammates' actions. The environment evolves from the state at one time to the state at the next time with an action possibly being taken during the time interval, saving only the current environment states. Each agent maintains knowledge of the environment in its KB, updating this knowledge as needed to carry out its plan or provide information to team members.

4. Agent Observability. To express agent observability, we define a query function CanSee(<observer> <observable> <cond>), where <observer> specifies the agent doing the observing, <observable> identifies what is to be observed, and <cond> specifies the conditions under which the <observer> can see the <observable>. When needed, the query is submitted to the knowledge base for evaluation after first forming the conjunction of the arguments. As <observablea> and <cond> may be predicates, missing values for variables will be supplied via unification if there are any such values that allow the <cond> to be satisfied, or else return FALSE. This allows an agent, for example, to determine the location (through variables) of a target if the conditions are satisfied (e.g., the target is within range). Time is implicit in this query and is taken to be the time of the current step. Note that strong constraints weaken agents' observability; weak constraints strengthen agents observability. The strongest constraint is FALSE, which means that the agent can see nothing. The weakest constraint is TRUE, which means that the agent can see everything.

Successful teamwork requires interdependency among the agents [6]. This suggests that an agent should know at least some things about what other team members can see. However, an agent may not know for sure that another agent can see something. Rather, an agent may only believe, based on its current beliefs, that another agent can see something. We then use BelieveCanSee(<believer> <observer> <observable> <cond>) to mean that one agent believes another agent can see something under certain condition.

We also make the assumption of "seeing is believing". While philosophers may entertain doubts because of the possibility of illusion, common sense indicates that, other things being equal, one should believe what one sees [13, 2]. Thus, we assume that an agent believes an observed property persists until it believes the property has been negated later.

In the following subsections, we describe the syntax and semantics of observability in more detail.

4.1. The Syntax of Observability. The syntax we use for observability is given in Table 4.1. For example, the observability specification for a carrier in the extended Wumpus World is shown below, where ca, rca, fi, rfi represent the carrier, carrier's detection radius, fighter and fighter's detection radius, respectively.

```
(CanSee ca (location ?o ?x ?y)
  (location ca ?xc ?yc) (location ?o ?x ?y)
  (inradius ?x ?y ?xc ?yc rca)
) // The carrier can see the location property of an object.

(CanSee ca (DO ?fi (shootwumpus ?w))
  (play-role fighter ?fi) (location ca ?xc ?yc) (location ?fi ?x ?y)
  (adjacent ?xc ?yc ?x ?y)
) // The carrier can see the shootwumpus action of a fighter.

(BelieveCanSee ca fi (location ?o ?x ?y)
  (location fi ?xi ?yi) (location ?o ?x ?y)
  (inradius ?x ?y ?xi ?yi rfi)
) // The carrier believes the fighter is able to see the
  // location property of an object.
```

TABLE 4.1
The Syntax of Observability

1:	$< observability >$	$:= (CanSee < viewing >)*$
2:		$(BelieveCanSee < believer >< viewing >)*$
3:	$< viewing >$	$:= < observer >< observable >< cond >$
4:	$< believer >$	$:= < agent >$
5:	$< observer >$	$:= < agent >$
6:	$< observable >$	$:= < property > \mid < action >$
7:	$< property >$	$:= (< property - name >< object >< args >)$
8:	$< action >$	$:= (DO < doer > (< operator - name >< args >))$
9:	$< object >$	$:= < agent > \mid < non - agent >$
10:	$< doer >$	$:= < agent >$

```
(BelieveCanSee ca fi (DO ?f (shootwumpus ?w))
  (play-role fighter ?f) (  ?f fi) (location ca ?xc ?yc)
  (location fi ?xi ?yi) (location ?f ?x ?y)
  (inradius ?xi ?yi ?xc ?yc rca) (inradius ?x ?y ?xc ?yc rca)
  (adjacent ?x ?y ?xi ?yi)
) // The carrier believes the fighter is able to see the
  // shootwumpus action of another fighter.
```

An agent has two kinds of knowledge, shared team knowledge, encoded in MALLET, and individual knowledge, contained in its knowledge base. The syntax of observability can be used either, as rules in an agent's knowledge base [31], or as capability incorporated into MALLET. In this paper, we encode observability as rules in agents' knowledge bases.

4.2. The Semantics of Observability. To give operational semantics to observability, we need to clarify the relationships of: 1) what an agent can see, what it actually sees, and what it believes from its seeing; 2) what an agent believes another agent can see, what it believes another agent actually sees, and what it believes another agent believes from its seeing.

In order to properly discuss the semantics, we need to introduce a notion of time, as preconditions and effects refer to different points in time. For purposes of exposition, we will simply assume that time is a discrete and indexed in order by the natural numbers, and use the indices to reference points in time. Since we are dealing with multiple agents, multiple actions may occur at the same time instant. We do not try to elaborate further on time in this paper, as there are a number of useful different ways of dealing with issues such as the synchronization among team members performing actions, and they are not central to the point of the paper.

Let $See_t(a, \psi)$ express that agent a observes ψ at time t. There are two cases to consider, first where ψ is a property, and secondly, where ψ is an action. When ψ is a property, seeing ψ means determining the truth value of ψ, with unification of any free variables in ψ. If ψ is an action, seeing ψ means that the agent believes the doer believed the precondition of ψ immediately before the action occurred and the doer believes the effect of ψ immediately after performing the action. We use the meta-predicate $Hold_t$(c) to mean c holds in the world (environment simulation) at time t. We make the assumption below:

$$\forall a, \psi, c, t, CanSee(a, \psi, c) \wedge Hold_t(c) \rightarrow See_t(a, \psi) \tag{4.1}$$

which means that if the condition c holds at time t and agent a has the capability to observe ψ under condition c, then agent a actually does determine the truth-value of ψ at time t.

Next, we consider the relation between seeing something and believing it. Belief is denoted by the modal operator BEL and for its semantics we adopt the axioms K, D, 4, 5 in modal logic. The assumption of "seeing is believing" is again stated separately for properties and actions. In the case of properties, it is formalized in the axiom below:

$$\forall a, \varphi, t, See_t(a, \varphi) \rightarrow [Hold_t(\varphi) \rightarrow BEL_t(a, \varphi)] \wedge [\neg Hold_t(\varphi) \rightarrow BEL_t(a, \neg\varphi)] \tag{4.2}$$

which says that for any property φ seen by agent a, if φ holds, agent a believes φ; if φ does not hold, agent a believes not φ ($\neg\varphi$).

Agent a's belief is more complex when an action, ϕ, is observed. Let $Doer(\phi)$, $Prec(\phi)$, $Efft(\phi)$ denote the doer, the precondition, and the effect of action ϕ. When agent a sees action ϕ performed by some agent, agent a believes that the agent believed the precondition and believes the effect. This process is expressed by the following axiom:

$$\forall a, \phi, t, See_t(a, \phi) \rightarrow BEL_t(a, BEL_{t-1}(Doer(\phi), Prec(\phi))) \wedge BEL_t(a, BEL_t(Doer(\phi), Eff_t(\phi))) \tag{4.3}$$

From the belief update perspective in our current implementation where beliefs are assumed persistent, for any $p \in Prec(\phi)$, agent a believes that $Doer(\phi)$ still believes p at time t (i. e. $BEL_t(a, BEL_t(Doer(\phi), p))))$ unless $\neg p$ is contained in Efft(ϕ). This is similar for *BelieveCanSee*.

An agent's belief about what another agent sees is based on the following axiom:

$$\forall a, b, \psi, c, t, t', BelieveCanSee(a, b, \psi, c) \wedge BEL_t(a, BEL_{t'}(b, c)) \rightarrow BEL_t(a, See_{t'}(b, \psi)) \tag{4.4}$$

which means that if agent a believes that agent b is able to observe ψ under condition c, and agent a believes c at time *t'*, then agent a believes at time t that agent b saw (*t'*<t), sees (*t'*=t), or will see (*t'*>t, which requires some prediction capability for agent a) ψ at time *t'*. In our approach, each agent focuses on the reasoning about current observability, not in the past or in the future. Therefore, the axiom above can be simplified as follows:

$$\forall a, b, \psi, c, t, BelieveCanSee(a, b, \psi, c) \wedge BEL_t(a, c) \rightarrow BEL_t(a, See_t(b, \psi)) \tag{4.5}$$

Note that agent a evaluates condition c according to its own beliefs.

Combining this with the previous assumption that "seeing is believing". we extend this to belief. We have two separate cases for properties and actions. When agent a believes agent b sees a property φ, a believes that b believes φ:

$$\forall a, b, \varphi, t, BEL_t(a, See_t(b, \varphi)) \rightarrow BEL_t(a, BEL_t(b, \varphi)) \tag{4.6}$$

When agent a believes agent b sees an action ϕ, a believes that b believes the doer believed the precondition at the previous time step and believes the effect at the current time step. This consequence is expressed by the following:

$$\forall a, b, \phi, t, BEL_t(a, See_t(b, \phi)) \rightarrow BEL_t(a, BEL_t(b, BEL_{t-1}(Doer(\phi), Prec(\phi)))) \wedge BEL_t(a, BEL_t(b, BEL_t(Doer(\phi), Efft(\phi)))) \tag{4.7}$$

4.3. Belief Maintenance. From the semantics, agents' observability is closely tied to their beliefs about the environment and other agents. Agents must update these beliefs when they perform, or reason about others', observation.

4.3.1. Maintaining Belief About Self's Observability. The axiom of "seeing is believing" bridges the gap between what an agent sees and what it believes. An agent maintains its beliefs in two aspects: 1) for an observed property, the agent believes the property; 2) for an observed action, the agent believes that the doer believed the precondition before the action and the doer believes the effect after the action. The algorithm for updating what an agent has observed, according to the observability rules, is given in Figure 4.1.

This algorithm builds beliefs in the believer's (i. e., agent self's), knowledge base by checking the following: Observing a property

- When evaluating observability `(CanSee self (<prop-name> <object> <args>) <cond>)`, self queries `<cond>` to environment KB. The query returns a list of substitutions of variables, or null if `<cond>` are not satisfied. When the returned tuple is not null, if the property holds in the environment, self updates its knowledge base with belief `(<prop-name> <object> <args>)` for each variable bindings, otherwise, self updates its knowledge base with belief `(not (<prop-name> <object> <args>))` for each variable bindings.

- Observing an action
  In the case of `(CanSee self (<action-name> Agd(≠ self) <args>) <cond>)`, the query `<cond>` is made with respect to environment KB as well. If the result of query is not null, self updates its beliefs by that self believes that agent Agd knew the precondition, and that Agd infers the effect. To handle the temporal issue correctly, self updates Agd's belief about the precondition first and then Agd's belief about the effect. These beliefs are useful in communication. For example, if agent a needs information I and believes agent b believes I, a may ask b for I.

updateSelfObs(self, KBself)
/* Let self be the agent invoking the algorithms. We denote the knowledge base for agent a by KB_a, for the environment by KB_{env}.*/
1: for each rule in KB_{self} of the form (CanSee self (prop object args) cond)
2: if cond is true in KB_{env} for some bindings of variables
3: if (prop object args) is true in KB_{env} for some bindings of variables
4: update(KB_{self}, (prop object args))
5: for each such binding of values to the variables;
6: else
7: update(KB_{self}, (not (prop object args)))
for each such binding of values to the variables;
8: for each rule in KB_{self} of the form (CanSee self (action doer args) cond),
if cond is true in KB_{env} for some binding of variables,
9: for each conjunct of precondition of action
10: update(KB_{self}, (BEL doer conjunct));
11: for each conjunct of effect of action
12: update(KB_{self}, (BEL doer conjunct));

FIG. 4.1. *An Algorithm of Maintaining Self s Belief by Direct Observation*

4.4. Maintaining Belief About Others' Observabilities. Figure 4.2 shows an algorithm for updating what an agent can determine about what other agents can see.

updateSelfBel(self, KB_{self})
1: for each rule of the form (BelieveCanSee self Ag (prop object args) cond) that
2: cond is true in KB_{self} for some binding of arguments to agents $Ag \neq self$
3: for each such binding of arguments to the variables
4: update(KB_{self}, (BEL Ag (prop object args)));
5: for each rule of the form (BelieveCanSee self Ag (action doer args) cond) that
6: cond is true in KB_{self} for some binding of arguments to agents $Ag \neq self$
7: for each conjunct of the precondition of action
8: update(KB_{self}, (BEL Ag (BEL doer conjunct)));
9: for each conjunct of the effect of action
10: update(KB_{self}, (BEL Ag (BEL doer conjunct)));

FIG. 4.2. *An algorithm of maintaining belief about others observabilities*

The algorithm records which agents are known to be able to see what, and updates what an agent believes, according to the precondition and effect of the actions it observes other agents performing. For the agent to determine whether a piece of information is needed by others, it simulates the inference process of others' observability to determine which is known by others.

- Observing a property
  In the case of `(BelieveCanSee self Ag(≠ self) <property> <cond>)`, a query `<cond>` is made with respect to KB_{self}. If the condition is satisfied, self believes Ag can see the property. However, self may or may not have knowledge of `<property>`. For example, a carrier may believe a fighter can smell a wumpus if the fighter is adjacent to the wumpus, but the carrier does not itself smell the wumpus.

- Observing an action In the case of (BelieveCanSee self Ag(≠self) (<action-name> doer(≠self) <args>) <cond>), <cond> is evaluated with respect to KB_{self}. Self adds tuples to KB_{self}, indicating that Ag believes that doer believed the preconditions of the action, and believes the effects of the action[1].

4.5. Execution Model. At each time step, every agent, denoted by self, has a function cycle: (possibly) observe, receive information from others, belief coherence, (possibly) send information to others, and act. If self needs an information item or produces an item needed by others, it will observe the world and other agents. It then checks messages and adjusts its beliefs for what it sees and what it is told. Self keeps track of the other agents' mental states by reasoning about what they see from observation, in order to decide when to assist the others with the needed information proactively. Finally, self acts cooperatively with teammates and enters the next time step.

An algorithm for overall belief maintenance along with the function cycle is shown in Figure 4.3. The algorithm begins with updateWorld by self's last action. We will not elaborate on how updateWorld works which is beyond the focus of this paper. Basically, the environment simulation updates the environment KB after receiving any action from the agent. Because the agent can infer the effect of its own action, the algorithm saves the effect as a new belief. UpdateSelfObs evaluates observability rules with information obtained from KB_{env} and updates KB_{self} with the results of the observation. UpdateSelfBel updates self's beliefs about what others' beliefs by observing environment and actions.

updateKB(self, action, KB_{self})
/* The algorithm is executed independently by each agent, denoted self below,
after the completion of each step in the plan in which the agent is involved.*/
1: updateWorld(action, self); //notify the environment to update KB_{env}
2: for each conjunct in the effect of action
3: update(KB_{self}, conjunct);
4: if self produces/needs information I
5: updateSelfObs(self, KB_{self}); //update KB_{self} by observability
6: updateSeflBel(self, KB_{self}); //update KB_{self} by beliefs about
//others observabilities
7: for each coming information I
8: update(KBself, I); //update KB_{self} by communication

FIG. 4.3. *An overall belief-maintenance algorithm*

The function update manages history and is responsible for coherence and persistence of belief in an agent's KB. The agent's beliefs about the world are saved as primitive predicates as they were expressed originally in the world. Such beliefs are generated from three sources: (1) belief from observation, i. e., a property self observes; (2) belief from inference, i. e., conjuncts inferred from the effect of the action self performs; (3) belief from communication, i. e., messages other agents send to self by communication. How does communication affect the agent's mental state? Van Linder et al. propose that the communication can also be translated to a belief saved in the mental state in the same way as observation is [13]. In any situation in which belief is required from multiple sources, conflicts may arise, such as self simultaneously sees $\neg\psi$ and hears ψ. A strategy is needed that prescribes how to maintain the coherence of the knowledge base of an agent in the case of conflicts among incoming information from different sources. Castelfranchi proposes that such a strategy should prescribe that more credible information should always be favored over less credible information [3]. To define a strategy complying with this idea, we propose that each source is associated with a credit and the credit decreases in this order: source from observation, source from inference, and source from communication. At certain time point, when an agent gets conflict information from different sources, it always believes what it sees.

Since the number of time steps could be infinite, an agent keeps only current beliefs in its mental state, except that the most recent one is kept, even if it is not generated currently. That an agent does not directly observe or infer some predicates from current observation does not mean it does not believe them. The agent has memory of them from before. Memory is useful in proactive communication; thus, if a piece of information is infrequently changed, at the time when agent a realizes that agent b needs the information, even if agent a does not have the information, agent a can tell agent b the information in its memory.

[1]Note, however, that self does not necessarily know what there values are. This is useful, however, in case self needs to make an activeAsk.

5. Proactive Communication. The purpose of proactive communication is to reduce communication overhead and to improve the efficiency or performance of a team. In our approach, proactive communication is based on two protocols named proactiveTell and activeAsk. These protocols are used by each agent to generate inter-agent communications when information exchange is desirable. Proactive communication answers the following questions pertinent to agent proactivity during teamwork. First, when does an agent send the information to its teammates if it has a new piece of information (either from performing an action or observing)? A simple solution could be sending the information when requested. That is, the agent would only send the information after it has received a request from another agent. Our approach is that the agent observes its teammates, and commits to proactive tell once it realizes that one of the teammates needs the information to fulfill its role and does not have it now. Meanwhile, if the agent needs some information, it does not passively wait for someone else to tell it; it should ask for this information actively. Second, what information is sent in a session of information exchange? There are two kinds of information that can be communicated. One is the information explicitly needed by an agent to complete a given plan, i. e., conjuncts in a precondition of plans or operators that the agent is going to perform. The other is the information implicitly needed by the agent. For example, if agent a needs predicate p and knows p can be deduced from predicate q, even if the providing agent does not know p, it still can tell agent a about q once it has q, because it knows that agent a can deduce p from q. This paper, however, deals only with agents communicating information that is explicitly needed.

The proactiveTell and activeAsk protocols are designed based on following three types of knowledge:

- Information needers and providers. In order to find a list of agents who might know or need some information, we analyze the preconditions and effects of operators and plans and generate a list of needers and a list of providers for every piece of information. The providers are agents who might know such information, and the needers are agents who might need to know the information.
- Relative frequency of information need vs. production. For any piece of information I, we define two functions, f_C and f_N. $f_C(I)$ returns the frequency with which I changes. $f_N(I)$ returns the frequency with which I is used by agents. We classify information into two types: static[2] and dynamic. If $f_C(I) \leq f_N(I)$, I is considered static information; if $f_C(I) > f_N(I)$, I is considered dynamic information. For static information we use proactiveTell by providers, and for dynamic information we use activeAsked by needers[3].
- Beliefs generated after observation. Agents take advantage of these beliefs to track other team members' mental states and use beliefs of what can be observed and inferred to reduce the volume of communication. For example, if a provider believes that a needer sees or infers information I, the provider will not tell the needer.

An algorithm for deciding when and to whom to communicate for activeAsk and proactiveTell[4] is shown in Figure 5.1.

Considering the intractability of general belief reasoning [7], our algorithm deals with beliefs nested no more than one-layer. This is sufficient for our current study on proactive behaviors of agents, which focuses on peer-to-peer proactive communication among agents. For activeAsk, an agent requests the information from other agents who may know it, having determined it from the information flow. The agent selects a provider among agents who know I and ask for I. For proactiveTell, the agent tells other agents who need I. An agent always assumes others know nothing until it can observe or reason that they do know a relevant item. Information sensed and beliefs about others' sensing capabilities become the basis for this reasoning. First, the agent determines what another agent needs from the information flows. Second, the observation rules are used to determine whether or not one agent knows that another agent can sense the needed information.

6. Empirical Study. While one would think that if one gives an agent additional capabilities, its performance would improve, and indeed this turns out to be correct, there are several other interesting aspects of our scheme to evaluate. For example, when there are several different capabilities, the interesting question arises of how much improvement each capability gives and which capabilities are the most important to add in different situations. Moreover, while it is obvious that one should not see decreasing performance from increasing

[2]Here, static information includes not only the information never changed, but also the information infrequently changed but frequently needed.

[3]In future work, we will address some statistical methods to calculate frequencies and hence will be able to provide more comprehensive proactive communication protocols.

[4]Note that there is no need to say anything anout previous time points, as those would have been handled when they were first entered. Furthermore, these is no need to consider $\neg I$ explicitly; if ture, it will be entered as a fact on its own.

activeAsk(self, I, KB_{self}, T)
/* Let T be the time step when the algorithm is executed.
Independently executed by each agent (self) when it
needs the value of information I.*/
1: candidateList=null;
2: if (I is dynamic and (I t) $\vee$ ($\neg I$ t) is not true in KB_{self} for any t$\leq$T)
3: if there exists a x$\geq$0 such that
4: ((BEL Ag I T-x) $\vee$ (BEL Ag $\neg I$ T-x)) is true in KB_{self}
5: let xs be the smallest such value of x;
6: for each agent Ag$\neq$self
7: if ((BEL Ag I T-xs) $\vee$ (BEL Ag $\neg I$ T-xs)) is true in KB_{self}
8: add Ag to candidateList;
9: randomly select Ag from candidateList;
10: ask Ag for I;
11: else
12: randomly select a provider
13: ask the provider for I;

proactiveTell(KB_{self}, T)
/* Independently executed by each agent (self), after it executes updateKB.*/
14: for each conjunct I for which (I, T) is true in KB_{self} and I is static
15: for each Agn needers
16: if (BEL Agn I T) is not true in KB_{self}
17: tell Agn I;

FIG. 5.1. *Proactive Communication Protocols*

capabilities, there are still interesting questions of how much performance increase can be obtained and how one can incorporate the capabilities into the system in a computationally tractable manner. And, one there is an interest in how the scheme scales with the number of agents involved. Our empirical study is intended to address these questions.

To test our approach, we have extended the Wumpus World problem [19] into a multi-agent version. The world is 20 by 20 cells and has 20 wumpuses, 8 pits, and 20 piles of gold. The goals of the team, four agents, one carrier and three fighters, are to kill wumpuses and get the gold. The carrier is capable of finding wumpuses and picking up gold. The fighters are capable of shooting wumpuses. Every agent can sense a stench (from adjacent wumpuses), a breeze (from adjacent pits), and glitter (from the same position) of gold. When a piece of gold is picked up, both the glitter and the gold disappear from its location. When a wumpus is killed, agents can determine whether the wumpus is dead only by getting the message from others, who kill wumpus or see shooting wumpus action. The environment simulation maintains object properties and actions. Agents may also have additional sensing capabilities, defined by observability rules in their KBs.

There are two categories of information needed by the team: 1) an unknown conjunct that is part of the precondition of a plan or an operator (e.g., "wumpus location" and "wumpus is dead"); 2) an unknown conjunct that is part of a constraint (e.g., "fighter location", for selecting a fighter closest to wumpus). The "wumpus location" and "wumpus is dead" are static information and the "fighter location" is dynamic information. Agents use proactiveTell to impart static information they just learned if they believe other agents will need it. For example, the carrier proactiveTells the fighters the wumpus' location. Agents use activeAsk to request dynamic information if they need it and believe other agents have it. For example, fighters activeAsk each other about their locations and whether a wumpus is dead.

We used two teams, Team A and Team B. Each team was allowed to operate a fixed number of 150 steps. Except for the observability rules, conditions of both teams were exactly the same. In the absence of any target information (wumpus or gold), all agents reasoned about the environment to determine their priority of potential movements. If they were aware of a target location requiring action on their part (shoot wumpus or pick up gold), they moved toward the target. In all cases, they avoided unsafe locations.

We report three experiments. The first explores how observability reduces communication load and improve team performance in multi-agent teamwork. The second focuses on the relative contribution of each type of

TABLE 6.1
Team Performance and Communication Frequency in Sample Run. T1: number of wumpuses left alive, T2: amount of gold left unfound, T3: total number of avtiveAsks used, T4: total number of proactiveTells used, T5: average number of activeAsks per wumpus killed, T6: average number of proactiveTells per wumpus killed

	$T1$	$T2$	$T3$	$T4$	$T5$	$T6$
$TeamA$	4.8	7.2	77.4	33.8	5.09	2.23
$TeamB$	15	14.6	67.6	28.8	13.6	5.9

belief generated from observability to the successes of CAST-O as a whole. Finally, the third evaluates the impact of observability on changing communication load with increase of team size.

Two teams are defined as follows:

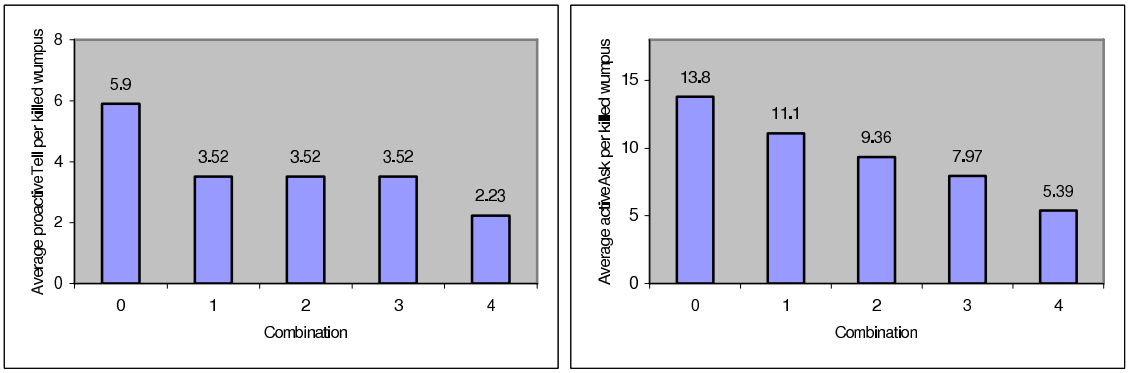
- Team A: The carrier can observe objects within a radius of 5 grid cells, and each fighter can see objects within a radius of 3 grid cells.
- Team B: None of the agents have any seeing capabilities beyond the basic capabilities described at the beginning of the section.

We use measures of performance, which reflect the number of wumpuses killed, the amount of communication used and the gold picked up. In order to make comparisons easier, we have chosen to have decreasing values indicate improving performance, e.g., smaller numbers of communication messages are better. To maintain this uniformity with some parameters of interest, we use the quantity not achieved by the team rather than the number achieved, e.g., the number of wumpuses left alive rather than the number killed. The experiments were performed on 5 randomly generated worlds. The results are shown in Table 1.

Table 1 shows that, as expected, Team A killed more wumpuses and found more gold than Team B. From other experiments we have learned that the further the agents can see, the more wumpuses they kill. It is interesting that the absolute number of communications is higher for Team A with observabilities than that of Team B, thus 33.8 vs. 28.8 for proactiveTell and 77.4 vs. 67.6 for activeAsk. The reason for the increased number of proactiveTells is that in Team A, the carrier, who is responsible for finding wumpuses and proactiveTelling wumpuses' locations to fighters, has further vision than that of the carrier in Team B. Hence the carrier in Team A can see more wumpuses. This feature leads to more proactiveTells from the carrier to the fighters in Team A. The number of proactiveTells can be reduced by the carrier's beliefs about the fighters' observability, i. e., if the carrier believes the fighters can see the wumpus' location, it will not proactiveTell the fighters. However, since the fighters' detect range is smaller than that of the carrier, the reduction cannot offset the number of extra proactiveTells. The reason for the increased number of activeAsks in Team A is that the more wumpuses they find, the more likely it becomes that messages are sent among fighters to decide who is closest to the wumpuses. Since fighters in Team A may find wumpuses by themselves, they need to ask other teammates if the wumpus is dead, to decide whether to kill it or not. Although the number of the messages could be reduced by factors such as allowing the fighter to see other fighters' locations and to see other fighters killing a wumpus, the increase cannot be totally offset because of the fighters' short vision. Hence, it makes more sense to compare the average number of messages per wumpus killed. In these terms, the performance of Team A, is much better than that of Team B, thus 2.23 vs. 5.9 for proactiveTell and 5.09 vs. 13.6 for activeAsk. Hence, our algorithms for managing the observability of agents have been effective.

The results of this experiment produced a bit of a surprise. By introducing observabilities to agents, the amount of communication actually increased slightly. This can be explained by the fact that because observability is a major means for an individual agent to obtain information about environment and team members; the more information obtained by the agent, the more messages were conveyed to help others. The proper way to interpret the results, then, is to normalize them by the performance of the team, which in this case is the average number of communications per wumpus killed, denoted by ACPWK, in this example. From this perspective, the amount of communication was reduced, as expected, also validating our approach.

6.1. Evaluating Different Beliefs Generated from Observability. The second experiment tested the contribution of different aspects of observability to the successful reduction of the communication. These aspects are belief about observed property, belief about the doer's belief about preconditions of observed action, belief about the doer's belief about effects of observed action and belief about another's belief about observed property. For simplify, we call them belief1, belief2, belief3 and belief4 correspondently. We test their contributions by



(a) proactiveTell protocol (b) activeAsk protocol

FIG. 6.1. *Average Communication Per Killed Wumpus in Different Combinations*

combining them. We used Team A and Team B in this experiment and kept all conditions the same as those of the first experiment. We used Team B, as reference to evaluate the effectiveness of different combinations of observability with Team A. We named this test combination 0, since there is none of such four beliefs involved in. For Team A, we tested another 4 combinations of these beliefs to show the effectiveness of each, in terms of ACPWK. These combinations are:

- Combination 0: Team B, which involves none of beliefs.
- Combination 1: In Team A, for each agent, leave off BelieveCanSee rules and do not process belief2 and belief3 when maintaining beliefs after observation. Therefore every agent only has belief1 about the world.
- Combination 2: Keep every condition in combination 1, except for enabling the belief2 process. This combination tests how belief2 improves the situation.
- Combination 3: Enabling the belief3 process in combination 2. This combination tests the effect of belief3.
- Combination 4: Add BelieveCanSee rules into combination 3. This combination tests the effect of belief4 as well as show effectiveness of the beliefs as a whole.

Each combination is run in the five randomly generated worlds. The average results of these runs are presented in Figure 6.1, in which one bar shows ACPWK for one combination.

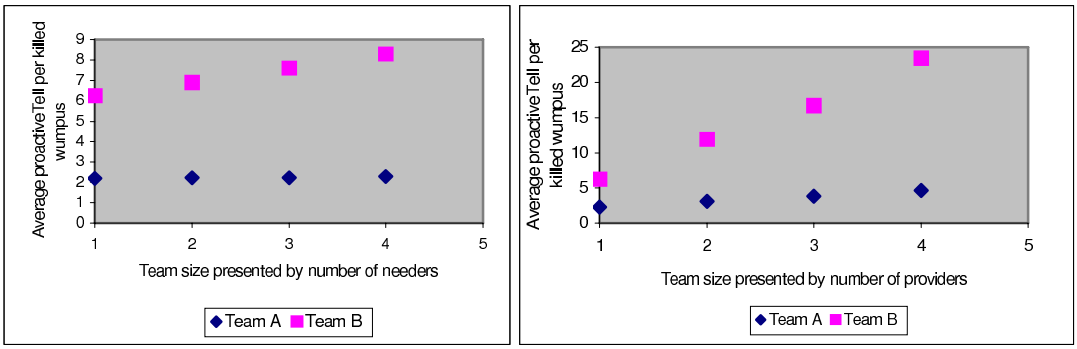
First of all that, agents' belief1 (combination 1) is a major contributor to effective communication, for both proactiveTell and activeAsk. For proactiveTell, in (a), compared to combination 0, ACPWK significantly drops from 5.9 to 3.52. For activeAsk, in (b), ACPWK drops from 13.8 to 11.1.

The second case, belief2 (combination 2) does not produce any further reduction and hence is not effective for proactiveTell, but produces improvement for activeAsk. For proactiveTell, when a provider sees an action, though it believes the doer knows the precondition and effect of the action, it does not know the precondition and effect by itself. So for this example belief2 can be of little help in proactiveTell. While for activeAsk, belief2 reduces ACPWK from 11.1 to 9.36, because with belief2, a needer will know who has a piece of information explicitly. Then it can activeAsk without ambiguity.

Third, for the same reason that belief2 only works for activeAsk, belief3 (combination 3) contributes little to proactiveTell but further decreases ACPWK to 7.97 for activeAsk.

Fourth, belief4 (combination 4) has a major effect on communications that applies to both protocols. It further drops ACPWK to 2.23 for proactiveTell and to 5.39 for activeAsk. Belief4 is particularly important for proactiveTell. For example, if the carrier believes that the fighters see a wumpus' location, it will not tell them.

This experiment examined the contribution of each belief deduced from observability to the overall effectiveness of communication. The result indicates three things. First, belief1 and belief4 have a strong effect on the efficiency of both proactiveTell and activeAsk. Therefore, CanSee/BelieveCanSee a property, the observability from which these two beliefs generated, can be generally applied to dual parts communication involving both Tell and Ask. Second, belief2 and belief3 have weak influence on the efficiency of proactiveTell, this suggests



(a) needers increment (b) providers increment

FIG. 6.2. *The Comparison of ProactiveTell with Different Team Size*

that CanSee an action may be applied to communication which incurs more Ask than Tell, such as goal-directed communication. Third, these beliefs work best together, because each of them provides a distinct way for agents to get information from the environment and other team members. Furthermore, they complement each other's relative weaknesses, so using them together better serves the effectiveness of the communication as a whole.

6.2. Evaluating the Effect of Observability Communication Load with Increased Team size. We designed the third experiment to show how communication load scales with increased team size. Based on the assumption that proactiveTell brings more communication into play than activeAsk, we choose to test the proactiveTell protocol. ActiveAsk is directed to only one provider at certain time, while the proactiveTell goes to all needers who do not have the information. If the test results are good for proactiveTell, we can expect that they are valid for activeAsk as well.

We used the same sensing capabilities for Teams A and Team B as in the first experiment. However, we increased the number of team members by 1, 2 and 3, in two tests that we ran. In the first test, we increased the number of needers, (i. e. fighters) and kept the same number of providers, (i. e. carriers). In the second test, we did it the other way around. In each test, for each increment and each team, we ran the five randomly generated worlds and used the average value of ACPKW produced in each world.

Figure 6.2 shows the trend of ACPKW as a function of increasing team size. In (a), Team B has an obvious increase in ACPKW with increasing the team size. However, Team A keeps the same ACPKW. The cause can be attributed to two factors: first, the amount of the increasing proactiveTells is held down because if the carrier believes the fighters can see wumpus, the carrier does not perform proactiveTell; second, the more fighters there are, the more wumpuses will be killed, which enlarges the numerator of ACPKW.

In (b), increasing the number of providers breaks the constant trend in Team A and shows an increased ACPWK. However, comparing this increase to that of Team B, it is a moderate number. In Team B, every provider increment means almost double the number of proactiveTells. The communication load increases because of duplicate proactiveTells of the same information by different carriers. For example, each carrier always provides the wumpus' location to fighters when observing a wumpus. The carriers lack an effective way to predict when a piece of information is produced and by whom, which is one of our main concerns of future work. This experiment shows that the team empowered with observability has a slower growth of ACPWK with increase of team size, which may indicate that observability will improve team scalability in some sense.

7. Conclusion. In this paper, we have presented an approach to dealing with agent observability for improving performance and reducing inter-agent communication. Each CAST-O agent is allowed to have some observability to see the environment, and to watch what others are doing inside its detection range. Based on the observation, the agent updates its knowledge base and infers what others may know at the current time. Reasoning about what others can see allows agents to decide whether to distribute information and to whom. We have proposed a proactive communication mechanism to confer some advantage to related team members for realizing team interaction and cooperation proactively also. We have conducted an in-depth empirical

evaluation in an extended Wumpus World, comparing the relative numbers of proactiveTell, activeAsk, and wumpuses killed for agent teams with and without observability.

A major point to the proactive communication approach with observabilities is that the underlying system that interprets the team plans of the agents does most of the work for handling the observation, inference and communication. This need only be designed once. It is re-used as one moves from one domain to another. It is only the explication of the observability conditions that changes from one domain to another, and this is essentially linearly proportional to the number of agents and "size" of the domain properties that are to be observed.

Though currently we are considering just the times of information production or need, the same approach can be extended to uncertainty in observability as well. Additionally, our present proactive information algorithm analyzes the pre-conditions and effects of operators for which each agent is responsible in the team plan. The purpose of doing so is to determine potentially useful information flow among agents. However, this approach is restrictive. We would like to make the recognition of needed information more dynamic. One way to solve this problem is to recognize the plans of other agents by observing actions of the other agents, and tracking the sequence of sub-goals on which they are working dynamically. Using this information together with the action an agent has most recently performed, the most likely information needs of other agents can be dynamically estimated over a finite time horizon. Then we can send other agents only unknown information that will be needed in the near future.

Acknowledgement. This work was supported in part by DoD MURI grant F49620-00-I-326 administered through AFOSR.

REFERENCES

[1] T. Balch and R. C. Arkin, *Communication in reactive multi-agent robotic systems,*, Autonomous Robots, 1 (1994), pp. 27–53.
[2] J. Bell and Z. Huang, *Seeing is believing*, in Proceedings of Common Sense 98, 1998, pp. 391–327.
[3] C. Castelfranchi, *Guarantees for autonomy in cognitive agent architecture*, Intelligent Agents, (1996), pp. 56–70.
[4] P. R. Cohen and H. J. Levesque, *Teamwork*, Nous, Special Issue on Cognitive Science and Artificial Intelligence, 25 (1991), pp. 487–512.
[5] P. J. Gmytrasiewicz, E. H. Durfee, and D. K. Wehe, *A decision-theoretic approach to coordinating multi-agent interactions.*, in Proceedings of 12th International Joint Conference on Artificial Intelligence, 1991.
[6] B. J. Grosz, *Collaborating systems*, AI Magazine, 17 (1996).
[7] J. Y. Halpern and Y. A. Moses, *A guide to completeness and complexity for modal logics of knowledge and belief*, Artificial Intelligence, (1992), pp. 319–379.
[8] M. J. Huber and E. H. Durfee, *Deciding when to commit to action during observation-based coordination*, in Proceedings of the 1st International Conference on Multi-agent Systems, 1995, pp. 163–170.
[9] T. R. Ioerger and L. He, *Modeling command and control in multi-agent systems*, in 8th International Command and Control Research and Technology Symposium (ICCRTS), June 17-19 2003.
[10] H. Isozaki and H. Katsuno, *Observability-based nested belief computation for multi-agent systems and its normalization*, Intelligent Agent IV, (2000), p. LNAI 1757.
[11] G. A. Kaminka, D. V. Pynadath, and M. Tambe, *Monitoring deployed agent teams*, in Proceedings of International Conference on Autonomous Agents, 2001.
[12] G. A. Kaminka and M. Tambe, *Robust agent teams via socially-attentive monitoring*, Journal of Artificial Intelligence Research, 12 (2000), pp. 105–147.
[13] B. V. Linder, W. V. D. Hoek, and J. Meyer, *Seeing is believingâĂŤand so hearing and jumping*, Topics in Artificial Intelligence, LNAI 992 (1995), pp. 402–413.
[14] S. K. M. Fenster and J. S. Rosenschein, *Coordination without communication: Experimental validation of focal point techniques,*, in Proceedings of the 1st International Conference on Multi-agent Systems,, 1995, pp. 102–108.
[15] M. G. M. Genesereth and J. Rosenchein, *Cooperation without communication,*, tech. rep., Stanford Heuristic Programming project, Computer Science Department, Stanford University,, 1984.
[16] D. Musto and K. Konolige, *Reasoning about perception*, in Proceedings of the AAAI Spring Symposium on Reasoning About Mental States, 1993, pp. 90–95.
[17] D. Pynadath and M. Tambe, *Multiagent teamwork: Analyzing the optimality and complexity of key theories and models*, in Proceedings of the 1st Autonomous Agents and Multiagent System Conference, 2002.
[18] W. B. Rouse, J. A. Cannon-Bowers, and E. Salas, *The role of mental models in team performance in complex systems*, IEEE Transactions on Systems, Man, Cybernetics, 22 (1992), pp. 1296–1308.
[19] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, NJ: Prentice Hall, 2002.
[20] Y. Shoham and M. Tennenholtz, *On the synthesis of useful social laws for artificial agents societies (preliminary report)*, in Proceedings of the 9th National Conference on Artificial Intelligence, 1992.
[21] F. Stone and M. Veloso, *Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork*, Artificial Intelligence, 110 (1999), pp. 241–273.
[22] K. P. Sycara and M. C. Lewis, *Forming shared mental models*, in Proceedings of 13th Annual Meeting of the Cognitive Science Society, 1991, pp. 400–405.

[23] M. Tambe, *Towards flexible teamwork*, Journal of Artificial Intelligence Research, 7 (1997), pp. 83–124.
[24] A. D. Val, P. M.-R. II, and Y. Shoham, *Qualitative reasoning about perception and belief*, in Proceedings of 15th International Joint Conference on Artificial Intelligence, 1997, pp. 508–513.
[25] M. Viroli and A. Omicini, *An observation approach to the semantics of agent communication languages*, Applied Artificial Intelligence, 16 (2002), pp. 775–793.
[26] M. Wooldridge and N. R. Jennings, *Intelligent agents: Theory and practice*, The Knowledge Engneering Review, 10 (1995), pp. 115–152.
[27] M. Wooldridge and A. Lomuscio, *Multi-agent vsk logic*, Proceedings of the 17th Eurapean Workshop on Logics in AI, (2000).
[28] P. Xuan, V. Lesser, and S. Zilberstein, *Communication decisions in multi-agent cooperation: Model and experiments*, in Proceedings of the 5th international conference on autonomous agents, 2001, pp. 616–623.
[29] J. Yen, X. Fan, and R. A. Volz, *A theoretical framework on proactive information exchange in agent teamwork*, Artificial Intelligence Journal, 169 (2005), pp. 23–97.
[30] J. Yen, X. Fan, R. Wang, S. Sun, and R. A. Volz, *Context-centric needs anticipation using information needs graphs*, Journal of Applied Intelligence, 24 (2006), pp. 75–89.
[31] Y. Zhang and R. A. Volz, *Modeling cooperation by observation in agent team*, in Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC'05), 2005, pp. 536–541.

Edited by: Marcin Paprzycki, Niranjan Suri
Received: October 1, 2006
Accepted: December 10, 2006