Context-Centric Needs Anticipation Using Information Needs Graphs

Xiaocong Fan, Rui Wang, Shuang Sun, John Yen
School of Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16802
{zfan,rwang,ssun,jyen}@ist.psu.edu

Richard A. Volz
Department of Computer Science
Texas A&M University
College Station, TX 77843
volz@cs.tamu.edu

Keywords: Multi-Agent Systems, Teamwork, Information Needs, Contexts

Abstract

Effective agent teamwork requires information exchange to be conducted in a proactive, selective, and intelligent way. In the field of distributed artificial intelligence, there has been increasing number of research focusing on need-driven proactive communication, both theoretically and practically. Among these works, CAST has realized a team-oriented agent architecture where agents, based on a computational shared mental model, are able to anticipate teammates' information needs and proactively deliver relevant information to the needers in a timely manner. However, the first implementation of CAST takes little consideration of the dynamics of the anticipated information needs, which can change in various ways as the context develops. In this paper we describe a novel mechanism for organizing and managing the "context" of information needs. This allows agents to dynamically activate and deactivate information needs progressively. It has been shown that the two-level context-centric approach can enhance team performance considerably.

I. INTRODUCTION

Studies in cognitive science have shown that human team members tend to proactively share new information to achieve their joint goals [1], [2], [3]. "Proactive information delivery"—sharing relevant information without being asked—has been identified by some psychologists [4], [5], [6] as a key characteristic of effective human teams. This has been motivating researchers in Multi-Agent Systems field to also consider empowering software agents with the capability of proactive information delivery [7].

Proactive information delivery is important in multi-agent systems because effective teamwork relies on effective communication, which plays an essential role in dynamic team formation [8], in coordinating shared activities [9], [10], [7], and more theoretically, in the forming, evolving, and terminating of both joint intentions [11] and shared plans [12].

Researchers in cognitive science have tried to tie proactive information delivery to the capability of anticipating others' future information needs based on certain shared mental models among team members [13]. Yen, Fan, and Volz [14] have formalized the notion of information needs and investigated the classification of information needs typically emerging in agent teamworks. As they pointed, information needs can be derived from the knowledge of multi-agent information dependence [15] regarding future physical or epistemic commitments (i.e., activities or intentions). Although information dependence is very important for supporting social reasoning in multi-agent cooperations, the existing researches on dependence theory or framework (e.g.,

[16], [7], [17], [18]) have virtually ignored the issue of dependence dynamics, assuming the models of dependence relations are fixed once established. However, oftentimes, the notion of information needs is context-dependent: the collection of information an agent needs to consider may change over time. It thus indicates that agents in a multi-agent system ought to track the dynamics (contexts) of others' information needs in order to offer timely help without disturbing the recipients with information no longer relevant to their activities.

CAST (Collaborative Agent architecture for Simulating Teamwork) is a teamwork model focusing on understanding proactive information exchange among teammates based on shared teamwork processes [7]. In the first implementation of CAST [19], [7], information needs are derived from a fully decomposed shared team plan and stored in a data structure called information-flow-table (IFT). Such a context-insensitive management of information needs is significantly limited in effectively tracking the dynamics of others' information needs. And, as we mentioned above, this can introduce unnecessary overhead, which may heavily affect the performance of teams working under time pressure. For instance, agents may keep sharing irrelevant information while delaying or neglecting of sending what the receiver badly needs. Team performance could suffer more from the IFT management of information needs as team complexity increases; this is especially true for agent teams working in information intensive domains like Network-Centric Warfare [20].

To enable agents to communicate in a more intelligent and selective way, a better mechanism for managing the dynamic activation and deactivation of teammates' information needs is highly needed. The intent of this research is to explore and evaluate such a mechanism. The rest of the paper is organized as follows. We next clearly define the problem under our consideration. In Section II we give an overview of the CAST architecture. We set the stage in Section III by outlining our approach and giving an example for illustrating concepts in ensuing sections. The technical details of the context-centric mechanism and information-needs-graphs are covered in Section IV. In Section V we report the experiments of evaluating the new management mechanism. We compare our approach with the relevant work in the literature in Section VI, and finally conclude in Section VII.

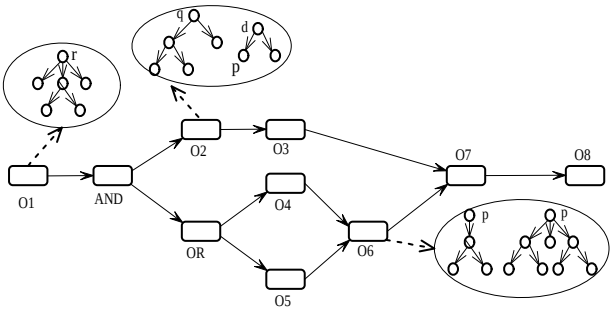

Fig. 1. Dependence changes as work proceeds

A. Problem Definition

Consider the abstract workflow shown in Fig. 1, where rounded boxes represent task-related operations or control-points (AND point, OR point). The activities involved in the branches leaving from an AND point can be done parallelly, and probably by different groups of actors, but they have to be finished before the actors can proceed to the common successive operation (e.g., $O7$ in Fig. 1) of all the branches. The actors entering an OR point can proceed to the common successive operation of the branches of the OR point whenever the activities in one of the branch have been completed.

It is a typical treatment in AI that an operation in a plan can be associated with certain constraints—some describes preconditions for executing the operation, some describes under what situation the execution must be terminated. Here, we assume each operation of a team process (e.g., Fig. 1) may have associated preconditions and termination conditions, and each OR point may have associated preference conditions (for dynamically choosing a workable branch). In Fig. 1, the information dependences derived from the constraints (represented as predicates) of an operation are clustered in an oval.

In complex situations, the collection of information dependence that an agent needs to consider can change as its activity proceeds. In particular, an agent no longer needs to consider dependence r associated with $O1$ after it is done (assuming all the other operations have nothing to do with r). When an agent chooses a specific branch of an OR point (e.g., $O5$), those dependences associated with the other branches (e.g. $O4$) should be out of concern. Of course, it would be worthwhile for an agent executing $O6$ to consider those dependences associated with $O2$ and $O3$

because another agent may be waiting for help (in this example, d depends on p). To complicate the issue further, each operation may be a complex subprocess in itself. Then, collecting the set of information dependences that reflects the current work progress could involve two-dimensional exploration. In this paper, we will introduce the notion of information-needs graphs as a means to organize and progressively activate/deactivate others' information needs.

In general, relevance is a matter of degree, and research on information relevance could be centered on utility relevance, semantic relevance, statistical relevance, etc. In this paper, however, we only consider logical relevance in the sense that whether or not a piece of information is relevant to the execution of a certain task (or the fulfillment of a commitment). In other words, we are concerned about whether or not a piece of information is useful in the evaluation of the constraints associated with a task. Because task constraints are represented as information needs in terms of first-order formulas, in the following descriptions, we also interchangeably say "information I' is relevant to information need N". We leave the degree of relevance open for future work. But we do consider both direct relevance and indirect relevance. Information $I(\vec{t})$ is *directly* relevant to need $N(\vec{x})$ iff $I \equiv N$ and the vector of constants $\vec{t}$ can be successfully unified with variables in $\vec{x}$. Information $I(\vec{t})$ is *indirectly* relevant to need $N(\vec{x})$ iff information regarding N can be derived from $I(\vec{t})$ together with other kinds of information using certain inference rules. For instance, suppose an agent use the information about enemy locations and moving directions to judge threat levels: $threatLevel(?e, High) \leftarrow inActiveZone(?e) \wedge Direction(?e, Close)$. If $threatLevel(?e, ?x)$ represents an information need of an agent A, then information like $inActiveZone(e12)$ is indirectly relevant to what A needs. When we say "irrelevant information (with respect to an agent)" we mean the information is in no way (neither directly nor indirectly) relevant to any of the agent's information needs, which, as we claimed above and address bellow, can change over time.

II. BACKGROUND: THE CAST SYSTEM

CAST is a teamwork model that enables software agents to effectively track teamwork processes, anticipate teammates' potential information needs, and proactively share information relevant of others' needs [7], [21]. Figure 2 shows the key components of the CAST agent architecture, where

- *Team process tracking* allows agents to monitor and adapt to the progress of team activities;

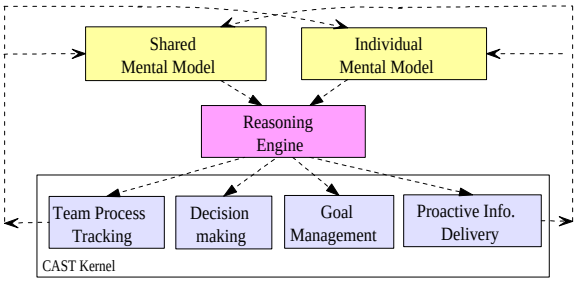

Fig. 2. The CAST Agent Architecture

- The *goal management* module is used to manage active goals being or to be pursued;
- The *decision-making module* is used to determine the next course of action and whether/how to help other teammates with their information needs;
- The *Reasoning engine* executes a standard sense/decide/act loop, initiate collaborations (e.g., dynamically recruiting team members) among teammates based on the shared mental state and individual mental state;
- The *Proactive Information Delivery* module is responsible for identifying opportunities to satisfy teammates' information needs;
- The *Individual Mental Model* stores those mental attitudes privately held by individual agents, including individual expertise, models of teammates, etc;
- The *Shared Mental Model* stores the knowledge and information that are shared by all team members, including team structures and team processes.

We next briefly describe how team process tracking, proactive information delivery, and the shared mental model are implemented in CAST. Details can be referred to [21].

A. Tracking Teamwork Process

In CAST, team plans, team structures and other teamwork knowledge are specified in MALLET—a Multi-Agent Logic Language for Encoding Teamwork [22]. A team plan is composed of preconditions, effects, termination conditions, and a procedural description of the process. The preconditions of a plan, represented by a logical conjunction, prescribe a necessary condition (pre-requisite resource and information) under which the plan can be performed. The effects, also represented by a logical conjunction, describe what must hold after the plan is done successfully.

Two types of termination conditions can be specified for a plan: success conditions (the plan should be terminated if the conditions become true), and failure conditions (the plan should be terminated if the conditions become false). The process of a plan describes the procedure of how a team accomplishes a task by using plan-invocation statements and constructs such as **seq** (sequential), **par** (parallel), **if** (conditional), **while** (iterative) **choice** (optional), etc. We will give an example team profile in Section III to show how team plans are structured.

To facilitate dynamic reasoning and monitoring, at compile time the embedded MALLET compiler transforms team plans into PrT nets (Predicate-Transition Nets), which are organized hierarchically so that agents can explore the team process at an appropriate level of details. Figure 3 illustrates how the hierarchical PrT nets work. In the net at the top layer the two places after transition $T6$ are currently holding task tokens. The sub-net between $T6$ and $T7$ are transformed from a parallel statement in the original team plan: the branch $workOnFire(?fire)$ is only firable for firefighter agents while the branch $rescueOnFire(?fire)$ is only firable for rescue agents. Suppose the net reflects the current mental state of a firefighter agent and the preconditions of $workOnFire$ are satisfiable wrt. the agent's belief base, the agent will fire the operation $workOnFire(?fire)$ by pushing the net at the top layer into an internal process-stack and transferring control to the net at the middle layer. Similarly, in case that the token configuration of the net at the middle layer is as shown in Figure 3, the firing of the operation $extinguishFire(?fire)$ will result in the transfer of control to the PrT net at the bottom layer.

CAST agents in a team are initially equipped with the same MALLET input, they thus will have the same set of PrT Nets, which establish a shared picture of the static process to be followed by team members. The team plans, accordingly the PrT nets, can be partial in the sense that certain roles (agent variables) need to be determined at run time. This to some extent is similar to the idea of incomplete recipes in the SharedPlans theory [23].

CAST agents are required to inform other teammates of their progress at critical points along the active team process. For instance, the agents to be involved in a team operator have to be all ready (by sending synchronization messages) before executing the operator; all the involved agents will be informed of the decision on the next course of action immediately after an alternative at a choice point has been chosen by some team member. By allowing agents to exchange information regarding their teamwork progress, they can have a global picture of the dynamic progress of the committed team activity. This enables agents to determine how their

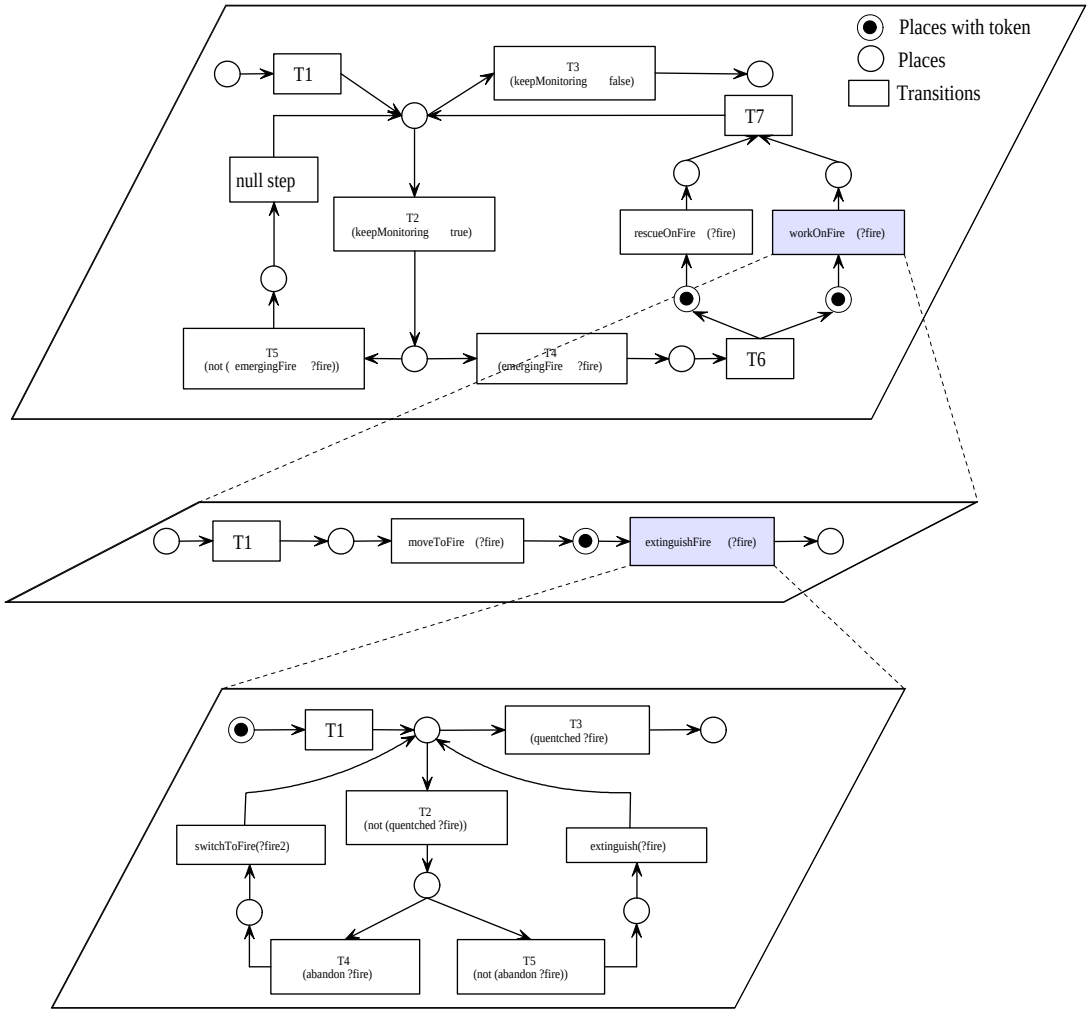

Fig. 3. Process Hierarchy–An Illustration

individual actions fit together, to act proactively to achieve coherent teamwork, and to anticipate the opportunities of offering help.

The hierarchical organization of team processes carefully separates domain independent aspects from domain dependent ones. This allows teamwork tracking to operate only at the needed level of detail.

B. Information Needs and Proactive Information Delivery

Information needs are represented as predicates annotated with a list of agent names; that means those agents need to be informed of the information described by the predicates. CAST

supports two types of information needs: action-performing information needs and plan-terminating information needs.

Prior to performing a plan or action, an agent typically needs to check whether the plan or action is both physically and epistemically feasible [24]. The epistemics feasibility is typically reflected in the preconditions of a plan or action. For this reason, the information needs generated from action preconditions are called *action-performing information needs*.

The execution of a MALLET plan may be terminated if (a) the expected effects (i.e., goal) of the plan is already achieved; (b) the execution of the plan cannot proceed further (i.e., failed in executing the embedded actions or plans); or (c) the execution of the plan becomes irrelevant (i.e., the goal is abandoned). This conforms to the Joint Intention theory [25], which requires that all the agents involved in a joint persistent goal (JPG) take it as an obligation to inform other agents regarding the achievement or impossibility of the goal. CAST agents respond to these three cases differently. In case (a), the agents simply proceed to pursue the next team activity; in case (b), the agents will try to choose some other alternative to re-attempt the goal; in case (c), the agents will switch their attentions to a new joint goal. To do this, all the agents should commit to letting others know whenever they detect a team activity is terminated or is about to terminate. The detection of plan termination is done by periodically checking the truth values of the effects and termination conditions specified for a plan. The information needs generated from plan effects and termination conditions are called *plan-terminating information-needs*.

The proactive information delivery behavior is realized in the DIARG (Dynamic Inter-Agent Rule Generator) algorithm [7], [19]. DIARG allows an agent to initiate communications without being asked based on a data structure called Information-Flow Table (IFT), which is a collection of tuples of the form *(info-needs, potential-info-needers, potential-info-providers)* generated offline by analyzing the shared team plans and the roles each agent can play in the plans. Whenever an agent acquires new information, the agent will check whether there is a match between the information and the needs in its IFT; if so, the agent will further decide whether to help the needers based on certain criteria. For instance, an agent will not send a piece of information to agent B if it is highly likely that agent B itself can observe the information. Also, no communication is carried out if the agent believes another agent, say C, has a better position to provide the information to B.

C. The Shared Mental Model

Both anticipation of information needs and proactive information delivery are enabled by the computational shared mental model implemented in CAST. The notion of shared mental models is a hypothetical construct that has been put forward to explain certain coordinated behaviors of human teams (e.g., [13]). A shared mental model not only establishes a mutual awareness among team members, it also requires the involved agents to commit to maintaining the mutual awareness. In other words, all the agents in a team should re-establish the "shared" status whenever disagreements occur. Joint intentions [25] are a special kind of shared mental models, where an agent, who has a joint intention with others, needs to inform them whenever it detects the goal is already achieved, becomes irrelevant or unattainable.

The shared mental model (SMM) implemented in CAST has four components: team structure, domain knowledge, information-flow table (IFT), and team processes. The *team structure* component captures those knowledge specifying team (sub-team) memberships, abstract roles, and agent-role relationships. The *domain knowledge* component contains domain independent (e.g., communication protocols) or dependent (e.g., domain expertise, inference rules) common knowledge. The observability of team members can also be shared as common knowledge to enable them to approximately reason about others' mental state. Shared team processes in the SMM of each CAST agent actually refer to the combination of two parts: the static structure and the dynamic progress (i.e., the token configuration of PrT nets) of team processes (cf. Section II-A), which together provide a common approximate understanding of the status of the committed team activity. As we have mentioned, IFT is a table-like structure that records information-flow relationships. The potential information needers and potential information providers of each entry in IFT may contain uncertainties (i.e., there may exist agent variables to be determined at run time); such uncertainties in the IFT can be resolved as the team proceeds along the shared processes.

III. Setting the Stage

The first implementation of CAST [7] adopted an IFT-based approach to manage teammates' information needs. The proactive information delivery behavior enabled by IFT is virtually similar to static *subscribe*: agents at compile time implicitly subscribe their information needs from potential providers, who then at run time proactively send the relevant information whenever

it becomes available. The IFT-based approach has exhibited at least two limitations. First, the IFT-based approach only supports naive matching. As team complexity scales up, there may exist dependence relationships among information needs (e.g., hierarchical decomposition of a high-level information needs into many lower-level ones) and such dependence knowledge may distributed among a group of agents. Being able to leverage the dependencies among information needs, an agent can share information at multiple levels, and even exchange information that only partially matches with teammates' information needs. This limitation has been addressed using the notion of distributed inference trees [21].

The second limitation of the IFT-based approach is that the notion of information needs is *per se* context-dependent, but IFT does not track the shift of contexts. The dynamics of information needs, in particular, may lie in at least three aspects:

1) A choice point (specified using the **choice** construct) in MALLET is composed of several branches (potential ways) to achieve the goal associated with the choice point. Intuitively, for a team process with embedded choice points, only those information-needs associated with the selected branches should be activated. The relevance of such information needs cannot be determined until after the involved agents have selected a branch;
2) The already executed part of a team process may never be re-invoked again. Thus, the information needs associated with those "inactive" part should be removed out of the agents' concern;
3) A team or an agent may dynamically select goals to pursue (e.g., agents may terminate or suspend the current process when facing unexpected difficulties). The information-needs considered in pursuing one goal can be very much different from those considered in another. The agents should switch their attentions as they switch their goals (processes).

Although it has been shown that proactive communication based on the IFT approach can enhance team performance [7], lacking the ability to effectively track the relevance of information needs could significantly degrade the scalability of CAST. Also, the context-insensitive management of information-needs offered by IFT may induce agents to satisfy others' information needs that re no longer relevant, while delaying or neglecting of sending information that is badly needed. Of course, the needers could dynamically *unsubscribe* their information needs when they are no longer relevant. However, this will bring in other communication cost in cases where agents

need to switch their attentions back and forth, or in cases where there is a large number of needers of the subscribed information.

To address this limitation, we introduce a richer data structure called Information-Needs-Graph (ING) in Section IV, and propose a context-centric approach (ING-based) so that agents can flexibly conduct k-depth forward anticipation of others' information needs by taking full advantage of the shared teamwork progress.

For illustration purpose, we next give a slice of the team profile for a fire-rescue domain. The complete profile includes specifications for team structure, operators, and domain-dependent inference rules, which are omitted here for irrelevance. The readers can refer to [22] for the details of the syntax and semantics of MALLET.

- (**plan** fightBuildingFire ()
  (**process**
  (**while** (**cond** (gameOver false))
    (**if** (**cond** (emerging-fire ?fire))
      (**do** wholeTeam workOnFire ?fire)
      (noOp)))))
- (**plan** workOnFire (?fire)
  (**process**
  (**seq**
    (**do** wholeTeam moveToFire ?fire) ;phase I
    (**par** ;phase II
      (**do** Ambulance-team rescuePeople ?fire)
      (**do** Fighter-team extinguishFire ?fire)))))
- (**plan** rescuePeople (?fire)
  (**term-cond** (people-alive ?fire 0)(out-of-control ?fire))
  (**process**
  (**while** (**cond** (not-quenched ?fire)(not-abandoned ?fire))
    (**if** (**cond** (need-escape ?fire))
      (abandon ?fire) ;else
      (rescue ?fire)))))

- (**plan** extinguishFire (?fire)
  (**process**
  (**choice**
    ((pref-cond (has-chemical false))
      (**do self** extinguishFireM1 ?fire))
    ((pref-cond (has-chemical true))
      (**do self** extinguishFireM2 ?fire))))
- (**plan** extinguishFireM1 (?fire)
  (**term-cond** (people-alive ?fire 0)(out-of-control ?fire))
  (**process**
  (**while** (**cond** (not-quenched ?fire)(not-abandoned ?fire))
    (**if** (**cond** (need-escape ?fire))
      (abandon ?fire) ;else
      (extinguish ?fire)))))

In this example, the fire-fighting team, composed of a subteam of firefighters and a subteam of ambulance units, needs to respond to emerging fires in a virtual city. An agent's response to a fire is divided into two phases: move to the place catching fire, and fulfill its role at the fire place. The two subteams work parallelly in phase II: firefighters extinguish fires while ambulance units try to rescue injured or trapped people from the buildings on fire. In phase II, to extinguish a fire, the firefighters can choose from two different ways, depending on whether the buildings on fire contain chemical materials that can release toxic gases. The team members will abandon the current fire and switch to another if there is no more people alive in the buildings or the fire is out of control.

IV. The Context-Centric Approach

To endow CAST agents with the ability to dynamically manage and effectively anticipate others' information needs, we employ a flexible mechanism leveraging the shared understanding of the teamwork progress. The basic idea is to reduce the amount of information needs an agent needs to pay attention to at a given time. There are two options to do so. The first way is to utilize the hierarchical PrT Net representation of teamwork processes to reason about the validity

of others' information needs. Particularly, the start transition (e.g., $T1$ in Figure 3) of a PrT net can be linked to a table that stores the information needs relevant to the termination conditions of the corresponding plan and the pre-conditions of all the sub-plans at the immediate next level (i.e., not considering the pro-conditions of the subplans of subplans). This can significantly reduce the amount of information needs an agent needs to monitor because the approach leads to a level-specific inference of teammates' information needs. However, the drawback is two-fold. First, process at level i has to be terminated if some process at level above i is terminated. Thus, an agent active at process-level i should consider all the plan-terminating information needs associated with process-levels above i. However, CAST uses a process-stack to manage the trace of plan execution. To consider the information needs at a higher process-level, an agent has to establish logical links between PrT nets and sub-PrT nets. The existence of iterative statements in a team process will further complicate the reasoning.

Second, such a level-specific approach is too restricted in that it does not allow an agent to proactively help its teammates with their information needs at a deeper level (i.e., below i). Formally, at level i where P^i is the plan under concern, let $N^i(P^i)$ denote the set of information needs considered by a group G^i of agents who entered P^i (an information need n is in $N^i(P^i)$ iff n is a need of some agent in G^i with respect to plan P^i). Now, suppose G_i splits into two subteams G_A^{i+1} and G_B^{i+1}, who respectively trigger plans P_A^{i+1} and P_B^{i+1} at level $i+1$. We have the following relations:

Information needs considered by agents in G^i: $N^i(P^i)$,

Information needs considered by agents in G_A^{i+1}: $N^i(P^i) \cup N^{i+1}(P_A^{i+1})$,

Information needs considered by agents in G_B^{i+1}: $N^i(P^i) \cup N^{i+1}(P_B^{i+1})$.

Then, agents in G_B^{i+1} cannot help the needs of agents in G_A^{i+1}, and vice versa. As illustrated in Figure 4, forward anticipation at the same level reflects intra-team proactive information exchange (PIE), while in-depth anticipation reflects cross-team PIE. We thus need a mechanism that allows agents consider both intra-team and cross-team PIE. One important benefit of such a "look ahead" capability is to allow an agent not involved in executing a subplan to be able to help other teammates being involved in the subplan regarding their information needs. It is worthwhile to point out, however, that a "k-level look ahead" activation of information needs does not include activating those information needs associated with subplans in a branch of a choice point. The information needs associated with a choice branch will be activated only after

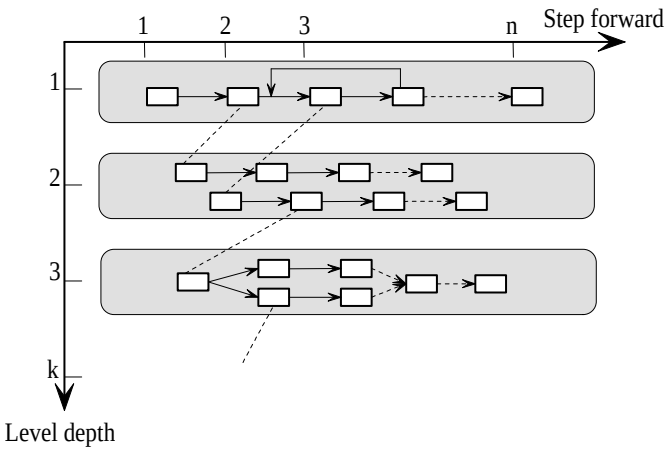

Fig. 4. 2-dimensional activation of information needs

the branch is selected.

To avoid the above limitations, we propose a context-centric needs activation/deactivation mechanism, which uses a tree-like structure-information needs graph (ING) to manage the activation/deactivation of information needs. ING captures the expansion of the task structure at every level. However, an ING is not simply a re-organization of the whole collection of PrT nets; rather, it offers a data structure that allows agents to effectively manage the dynamics of teammates' information needs at desired levels (by tracking the temporal relations of information needs). Even though INGs and PrT nets are separate and used for different purposes (INGs are used for managing the contexts of information needs while PrT nets are used for tracking the teamwork progress to better coordinate team members' behaviors), they do have run-time connections. The reasoning engine of CAST ensures that the dynamic status of teamwork processes be reflected in the corresponding INGs. It is worth paying this extra cost because knowing the current focuses of team members will help an agent refine the validity of their information needs so that it can better serve their needs and avoid initiating unnecessary inter-agent interactions as far as it can.

In the rest of this section, we explain the data structure and the algorithms for collecting active information needs of teammates.

A. The Generation of INGs

An ING may involve four kinds of nodes. A P-node, W-node, C-node, and H-node represent a plan invocation, an iterative statement, a choice statement, and a parallel statement in a team plan, respectively. Two relations are defined over these nodes: *sub-plan* relation and *sibling-plan* relation. We use P_p to denote a P-node corresponding to plan p. An ING is generated by transforming team plans according to the following rationales:

1) An iterative statement with index i (counted start from 0 according to the order of occurrence) in plan p is represented as a W-node W_p^i . We choose to explicitly represent the iterative points because this facilitates the reasoning regarding when to deactivate those information needs inside a loop.
2) A choice statement with index j (counted start from 0 according to the order of occurrence) in plan p is represented as a C-node C_p^j . Each choice point can specify several potential ways to achieve a goal. Those information-needs emerging from the selected branch should be activated as soon as a team or an agent makes a choice. The explicit representation of choice points in an ING allows agents to selectively activate those information needs associated with the chosen branch without necessarily considering those needs within other branches.
3) A parallel statement with index k (counted start from 0 according to the order of occurrence) in plan p is represented as an H-node H_p^k . We choose to explicitly represent the parallel points because this offers the flexibility that an agent can choose whether to consider the information needs of other sub-teams that are performing activities different from what the agent's own sub-team is pursuing.
4) Sequential statements (**seq**) and conditional statements (**if**) in plan p are captured by the *sibling-plan* relation. Two nodes $N1$ and $N2$ have a *sibling-plan* relation means the corresponding statement of $N1$ will be executed immediately before the corresponding statement of $N2$. Both nodes that correspond to the two alternatives of an **if** statement have *sibling-plan* relations with the nodes corresponding to the statements before and after the **if** statement. This is because which alternative will be chosen can only be determined when the execution reaches that point.
5) A plan-node P_p has *sub-plan* relations with each of the W-nodes W_p^i, C-nodes C_p^j , H-

nodes H_p^k, and all the P-nodes corresponding to plan-invocations in plan p but not within the scope of any iterative, choice, or parallel statements in p.

6) Each P-node is associated with information about (a) the potential or assigned doers of the corresponding plan, (b) the pre-conditions for performing the plan, and (c) the termination conditions of the plan. Both the preconditions and termination conditions can be organized as an AND/OR tree-like structure to endow CAST agents with the ability to infer teammates' indirect information needs.

7) Each C-node is associated with information regarding the preference conditions specified for the branches of the corresponding choice statement. The preference conditions can also be organized as an AND/OR tree structure to facilitate another level of reasoning.

At compile time , an ING is generated for each top-level team plan specified in MALLET (this is done in parallel with the generation of PrT nets). To generate an ING, starting from a top-level team plan tp, the ING generator first creates a P-node for tp, and then applies the algorithm $unfoldNode()$ recursively to each P-node to be unfolded. To terminate this generation process, a P-node will be skipped if the same P-node is already occurred in the trace along *sub-plan* relations from the root node (this occurs when a plan is invoked recursively).

Before giving the algorithm for generating INGs, we define two functions. Let function $lastPlans(S)$ be defined as:

$$\begin{cases} S & \text{if } S \text{ is a plan invocation, or a } \textbf{while } (\textbf{choice}, \textbf{parallel}) \text{ statement,} \\ lastPlans(S_n) & \text{if } S = (\textbf{seq } S_1 \cdots S_n), \\ lastPlans(B_1) \cup lastPlans(B_2) & \text{if } S = (\textbf{if } (cond\ .)\ B_1\ B_2). \end{cases}$$

Function $firstPlans(S)$ be defined as:

$$\begin{cases} S & \text{if } S \text{ is a plan invocation, or a } \textbf{while } (\textbf{choice}, \textbf{parallel}) \text{ statement,} \\ firstPlans(S_1) & \text{if } S = (\textbf{seq } S_1 \cdots S_n), \\ firstPlans(B_1) \cup firstPlans(B_2) & \text{if } S = (\textbf{if } (cond\ .)\ B_1\ B_2). \end{cases}$$

Algorithm 1: /*Generating ING from MALLET*/

unfoldNode(Node sn)

0. If sn is a P node, and a P node same to sn is already occurred in the trace: **return**
1. Let ls be the plan or statements referred to by node sn

2. $wPoints$:= get all the **while** statements in the scope of ls
3. $cPoints$:= get all **choice** statements in the scope of ls
4. $hPoints$:= get all **par** statements in the scope of ls
5. $subPlans$:= get all sub-plans invoked in the scope of ls
   excluding those in ($wPoints \cup cPoints \cup hPoints$)
6. If ($subPlans \cup wPoints \cup cPoints \cup hPoints == \emptyset$): **return**
7. For each sp in $subPlans$
   create a P-Node for sp and make it son of sn
8. For each **while** statement ws in $wPoints$
   create a W-Node wn for ws and make wn son of sn
   unfoldNode(wn)
9. For each **choice** statement cs in $cPoints$
   create a C-Node cn for cs and make cn son of sn
   unfoldNode(cn)
10. For each **par** statement ps in $hPoints$
    create a H-Node pn for ps and make pn son of sn
    unfoldNode(pn)
11. Establish sibling relations:
    if $(\mathbf{seq}\ \cdots S_1\ S_2 \cdots)$ occurs in the scope of ls
    the nodes corresponding to plans in $firstPlans(S_2)$ are siblings of the nodes corresponding to plans in $lastPlans(S_1)$
    if $(\mathbf{while}\ (cond\ .)\ S)$ occurs in the scope of ls
    the node representing this while statement is sibling of the nodes corresponding to plans in $lastPlans(S)$.

Figure 5 gives the ING generated by the ING generator for the fire-fighting example given in Section III.

Analytically, compared with the first approach discussed in the beginning of this section, this ING-based approach offers two additional benefits. First, the ING gives an abstract global picture of the temporal relations of information needs and puts teammates' information needs in an appropriate context that changes as the team process evolves. Second, as we already noted, if

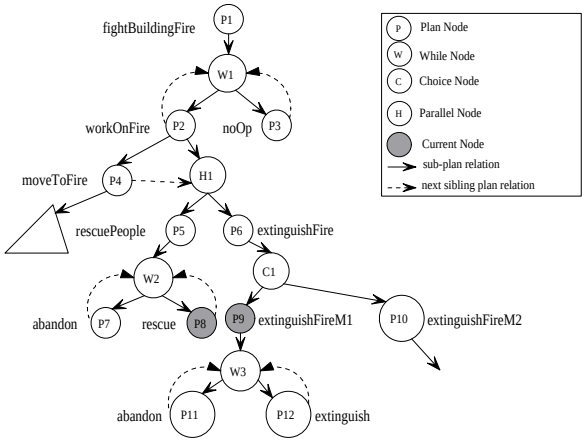

Fig. 5. An Example Info-Need Graph: Attention partitioned by work trace

a complex team plan can be decomposed into a task structure involving many layers of subplans, it will be too costly for an agent to monitor every pieces of information that may be needed by teammates at every levels of the task hierarchy. The ING-based approach offers the flexibility in achieving k-level in-depth anticipation. When k is set to 1, CAST agents will only consider those information needs at one process-level, without considering those in the embedded sub-processes. When k is set to $depth(ING)$, the depth of ING, CAST agents will consider all the information needs at every level of details. Such adjustable anticipatability can significantly enhance the scalability of CAST in anticipating and managing teammates' information needs.

B. Collecting Active Information Needs

INGs are generated offline but are used at run time to collect teammates' active information needs, leveraging the information about the current teamwork progress. For each ING, each agent dynamically maintains a collection of nodes marking the current work progress. The collection of 'working nodes" of an ING is constantly updated to reflect an agent's knowledge of the changing teamwork progress; the working nodes are then used as a context to temporally cut away inactive information dependences.

As shown in Fig. 5, the two filled nodes are the current working nodes, each reflecting the work progress of one of the two parallel activities. A 'working trace' of an ING refers to the path from the root node of the ING to a working node. For example, the path from node *fightBuildingFire*

to node *extinguishFireM1* is a working trace. An agent's attention about others' information needs can be partitioned by a working trace. Those information needs associated with the nodes (and the subtrees) on the left (wrt. the sibling relation) of a working trace are past attentions (e.g., the node *moveToFire* and its subtree); those on the right of a working trace are what should be considered. The existence of H-nodes in an ING means more than one subteam may work on different processes parallelly. In such cases, to encourage cross-team helping behaviors, those associated with all the son nodes (and the subtrees) of an H-node ought to be considered.

The collection of nodes that are under an agent's consideration (excluding those on the left of a working trace) can be further partitioned into three categories:

- "trace" nodes: those nodes appeared in the working trace;
- "uncertainty" nodes: those that are sons of a C-node. An uncertainty node needs to be explored later when it is actually chosen; and
- "to-do" nodes: all others except trace nodes and uncertainty nodes.

A node can change its status as team activities proceed: some uncertainty nodes can be activated, while some active nodes may become deactivated.

Given a working node, Algorithm 2 is used to collect teammates' active information needs relative to the current teamwork progress. Algorithm 2 adopts 1-level in-depth anticipation. The rationales implemented in Algorithm 2 are as follows.

1) Nodes left of the working trace are out of concern. This is reflected in clause 1 of $getRelevantNeeds()$, where the nodes before the trace node X along the sibling-plan relation are neglected;
2) An uncertainty node under a C-node is not considered until after the branch is chosen. This is reflected in Algorithm 2, where there is no treatment in clause 5 similar to clause 7. This is also reflected in procedure *getNodes()*, where in case P-Nodes are to be collected, the search is cut upon reaching a C-Node. Since there is no sibling relation among the sons of a C-node, no other branches except the chosen one on the work trace are taken into consideration. For instance, suppose in Figure 5 where an ambulance agent only considers the *rescue* node to be the current node. When collecting firefighters' information needs, the ambulance agent will not consider those associated with nodes *extinguishFireM1* or *extinguishFireM2* or their sub-nodes.

3) All the branches of a parallel statement are considered, this is reflected in clause 6 of $getRelevantNeeds()$. For the example in Figure 5, suppose in addition to knowing the progress of its own subteam (i.e., the ambulance team), an ambulance agent also knows the progress of the fire-fighting team (i.e., *extinguishFireM1* is the current node of the fire-fighting team). The ambulance agent can neglect firefighters' information needs associated with *extinguishFireM2* and its sub-nodes;
4) All the nodes in the scope of a **while** statement are considered (because they might be executed for an indefinite number of iterations); this is reflected in loop **B**.

Algorithm 2: /*Collecting Active Information Needs Using INGs */

getNodes(NodeList: *NL*, NodeType *NT*) **return** NodeList

List = $\emptyset$

Iterate X over Nodes in *NL*

IF X is of type *NY*

List.add(X)

IF X is a W-Node or H-Node

setN = $\{Y|Y$ is a Node of type *NT* accessible from X along *sub-plan* relation where only W-nodes or H-Nodes occur in the trace to $Y\}$

List.addAll(setN)

return List

getNeeds(NodeList: *NL*): **return** NeedsList

NList = $\emptyset$

$pNodes$ = getNodes(*NL*, *P-Node*)

NList.addAll($\cup_{P \in pNodes} pre(P)$) //$pre(P)$ returns the preconditions of P

$cNodes$ = getNodes(*NL*, *C-Node*)

NList.addAll($\cup_{C \in cNodes} pref(C)$) //$pref(C)$ returns the preference conditions of C

return NList

getNeedsA(NodeList: *NL*): **return** NeedsList

NList = $\emptyset$

$pNodes$ = getNodes(*NL*, *P-Node*)

NList.addAll($\cup_{P \in pNodes} pre(P)$) //$pre(P)$ returns the preconditions of P

NList.addAll($\cup_{P \in P \in pNodes} term(P)$) //$term(P)$ returns the termination conditions of P

$cNodes$ = getNodes(*NL*, *C-Node*)

NList.addAll($\cup_{C \in cNodes} pref(C)$) //$pref(C)$ returns the preference conditions of C

return NList

getRelevantNeeds(ING *ing*, Node *cur*): **return** NeedsList

Needs = $\emptyset$

traceNodes = getTraceNodes(*ing*, *cur*) //return all the nodes along the trace from $root$ up to cur

A: While $|$traceNodes$| > 0$ DO

X = traceNodes.removeFirst()

1. NL = getSiblings(X) //return all the siblings after X, excluding X
2. Needs.addAll(getNeeds(NL))
3. IF X is a W-Node: break
4. IF X is a P-Node

   Needs.add(term(X))
5. IF X is a C-Node

   Needs.add(pref(X))
6. IF X is an H-Node

   For each Node Y son of X except the one already in the trace being considered

   Cn = getCurNode(*ing*, Y)

6.1. Needs.addAll(getRelevantNeeds(subTree(*ing*,Y), Cn))

7 IF (X is a P-Node) and (X == cur)

   Needs.addAll(getNeeds(sonof(X)))

End {While}

B: While $|$traceNodes$| > 0$ DO //entering a loop

X = traceNodes.removeFirst()

8. NL = getSiblings( ) //return all the siblings of X, including X
9. Needs.addAll(getNeedsA(NL))

10. IF X is an H-Node
    For each Node Y son of X except the one already in the trace being considered
    Cn = getCurNode(*ing*, Y)
11. Needs.addAll(getRelevantNeeds(subTree(*ing*,Y),Cn))
12. IF (X is a P-Node) and (X == cur)
    Needs.addAll(getNeeds(sonOf(X)))
    End {While}
    return Needs

Here, procedure getNodes(NodeList: NL, NodeType NT) returns all the nodes of type NT in the list NL and all the nodes of type NT accessible from some node in NL along sub-plan relation where only W-nodes or H-Nodes occur in the trace. For example, getNodes([P4,H1],P-Node) will return {P4,P5,P6}, while getNodes([P4,H1],C-Node) will return $\emptyset$. Procedure getNeeds(NodeList: NL) returns all the information needs derived from the preconditions or preference conditions of P- or C-Nodes with respect to Node list NL. Procedure getNeedsA(NodeList: NL) is simply getNeed() with additional consideration of the termination conditions of P-Nodes.

Procedure getRelevantNeeds() includes two loops, which are almost the same except that the first loop deals with the case where a node is not within any WHILE statement while the latter deals with the case where a node is within the scope of a WHILE statement. In the first case, only those sibling nodes after the trace node need to be considered, but in the second case, all the sibling nodes (both before and after) of the trace node need to be considered.

C. Algorithm Analysis

Functionality Analysis: Algorithm 2 adopts 1-level in-depth anticipation. K-level ($k \geq 2$) in-depth algorithms can be similarly realized by extending $getNodes()$ such that sons of nodes in *NL* are considered up to k levels.

Algorithm 2 allows an agent in one sub-team to better anticipate the needs of agents in another sub-team if it is aware of the work progress of the other sub-team. This is reflected by the recursive call of *getRelevantNeeds( )* with the progress of each parallel branch (see clause 6.1 and clause 11). A natural way to let two (sub-)teams share their respective work progress is to designate a point of contact for each team. The two contact points then can proactively

inform each other their internal progress and share with their respective subteam members about the progress of other teams.

All the information needs emerging from within a loop will be considered until the termination of the loop. Hence, the less the scope of a WHILE statement, the better can gain from the algorithm. An agent would not benefit much from such context-centric needs anticipation if most part of the team process is within a loop. It seems that the idea of context-centric needs anticipation does not fit those processes that are a big loop in nature.

An agent could be involved in multiple teamwork processes in pursuit of multiple goals, and the agent may switch its attention from one goal to another. Because each top-level teamwork process has a corresponding ING, in such cases the agent can simply change the context from one ING to another. Such a flexible management of information needs has significantly increased the scalability of CAST in complex domains.

Complexity Analysis: Given an ING, suppose its average branching factor is b (this depends on the structure of the original team plans), the depth of the ING is d, and the depth of the current node is c.

The time complexity of finding all the siblings (clause 1) of a node is b. The time complexity of getting all the information needs for b nodes is $2b$ (clause 2). In the best case where the ING contains no H-nodes, the complexity of $getRelevantNeeds()$ is simply $O(c\cdot(b+2b)) = O(3bc)$.

The worst case is when all the trace nodes are H-nodes. Since an H-node will trigger a recursive call to $getRelevantNeeds()$, we can get the recurrence relation of $getRelevantNeeds()$:

$$a_d = c \cdot (b + 2b + (b-1)a_{d-1}),$$

that is

$$a_d = 3bc + (b-1)c \cdot a_{d-1}.$$

Solving this first-order linear recurrence, we have

$$a_d = M + MN \cdot (N^{d-2} - 1)/(N-1),$$

where $M = 3bc$, $N = (b-1)c$. Hence,

$$a_d \approx (MN^{d-2} - MN) \approx O(3bc(bc-c)^{d-2}).$$

The worst case complexity is $O(3b^{2d-2})$ when $b = c$.

The complexity in the real cases is much less than $O(3b^{2d-2})$, because it is rarely that all the trace nodes are H-nodes and c is less than b in most cases.

It is also worth noting that the relevance of information needs changes only when agents are executing sequential processes or choice points, or are informed of the team progress of other sub-teams. The algorithm is thus not supposed to be triggered at every step. Rather, it is triggered only upon knowing some critical teamwork progress has been made (e.g., an agent in one sub-team was told that another sub-team was reaching a choice point). Most of the information needs returned from an invocation of $getRelevantNeeds$ can be reused until the above situations occur. This not only avoids computation efficiency problem, but also makes sense for most complex domains, even though the collection of information needs considered by agents only approximately reflect the actual needs in between two invocations of the algorithm. An alternative is to develop iterative deepening depth-first algorithms [26] so that the in-depth anticipation can be conducted incrementally.

V. EVALUATION

We carried out a set of experiments to evaluate the context-centric approach for managing teammates' information-needs, comparing the performance difference of teams using ING-based mechanism and teams using IFT-based mechanism.

A. Experiment Design

Our experiments involved a team composed of three agents, two firefighters and one ambulance unit, which function in a simulated fire-rescue domain. Randomly emerging fires introduce tasks (goals) to the fire fighting team. The fire fighting team keeps working on a fire unless (a) the fire is already extinguished (i.e. the goal is achieved), (b) the fire becomes out-of-control (i.e., the goal becomes impossible), or (c) there is another more urgent fire emerging (e.g., there is no more people alive in the current fire, while there are people suffering in another fire. In this case, the current goal becomes irrelevant).

We assume communication cost is not neglectable. The execution of communication actions will lower down an agent's performance on its own task (e.g., delaying the execution of domain operations). However, other things being equal, we assume information is valued higher than communication cost. Thus, the experiments are designed such that if communication is done in

the right context at the right time, it can benefit the information receivers on their tasks, and this will ultimately affect the performance of the whole team.

Two types of critical information are of special interest to this experiment: I_1: *(out-of-control ?fireId)* and I_2 : *(people-alive ?fireId 0)*; both are plan-terminating information-needs. Both types of information are observable to the fire-fighting members, as long as they are near enough to the fire (i.e., within their sensing ranges). To introduce a change in contexts, we divide the response of the team to a fire into two phases: (1) moving near to the fire place, and (2) working on (extinguish fires or rescue people) the fire. We set the ambulance to move much faster than firefighters such that it is always the first in the team who knows information of type I_1 and I_2. In such a setting, the information-needs of firefighters change: in phase 1 they need information of both type I_1 and I_2, while in phase 2 they no longer need them from the ambulance because they can sense the information themselves.

To investigate the difference, we designed two proactive teams. They differ in the management of information-needs: one team uses the new context-centric approach (we call it team P), and one team uses the original IFT approach (we call it team S, since it's subscribe-based). During phase 1, the ambulances in both team P and S will proactively inform other team members whenever they sensed information of type I_1 or I_2 so that the whole team could save effort and time to work on other emerging fires; during phase 2, knowing the context has changed (fighters now can sense) the ambulance in team P will not proactively send dynamic information of type I_1 or I_2; while the ambulance in team S (insensitive to context) still tries to send the irrelevant information to fighters. To show the benefits of proactive communication against other communication strategies, we also designed team R (reactive team) and team N (non-communication team). Team R uses reactive strategy: during phase 1, the fire fighters will ask the ambulance unit in certain controllable frequency (e.g. initiate an ask every 10 "extinguish" operations) whether the fire is out-of-control or whether there is no people alive, and keep an eye on the replies from the ambulance unit so that they can shift their goals in a timely manner. Team N works even more inactive: there is no communication regarding information of type I_1 or I_2. The communication behaviors of these four team are listed in Table I.

The number of fires is chosen as the control variable. To simplify the domain complexity, fires pop up one after another: the next fire pops up immediately after the current fire becomes out-of-control, or there is no more people alive, or it is quenched. For each team we launched

TABLE I

COMMUNICATION BEHAVIORS

Team type	Phase 1	Phase 2
P team	inform I_1 and I_2 proactively	
S team	inform I_1 and I_2 proactively	inform I_1 and I_2 proactively
R team	ask/reply I_1 and I_2	
N team		sense I_1 and I_2

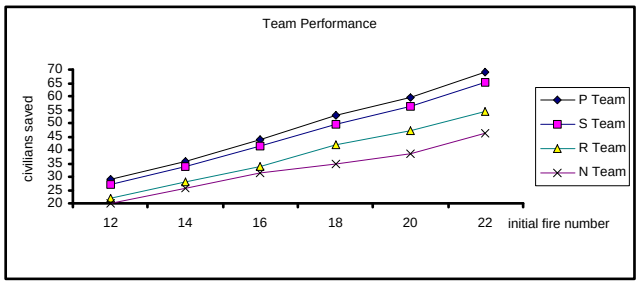

Fig. 6. Experiment I

300 runs with different domain configurations (randomizing on fire location, agent location, fire fierceness, number of people captured in fire). Figure 6 plots the experimental results, where team performance is measured in terms of the number of civilians saved.

B. Result Analysis

According to Figure 6, the performance of non-communication team (N team) is the worst. This is not surprising, since in this fire-fighting domain communication is crucial for team members to coordinate in task (goal) selection and to collaborate their behaviors.

The performance of R team is worse than the two proactive teams (P and S) because of the widely recognized limitations of reactive communication: an information consumer may not realize certain information it has is already out of date, or may not know when is the best time to request information. Thus before using a piece of information, this agent needs to verify the validity of the information, or needs to ask for a piece of information in a certain frequency. Consequently, the team can be easily overwhelmed by the amount of communications. Here, proactive teams outperformed reactive team because the burden of information-acquisition is

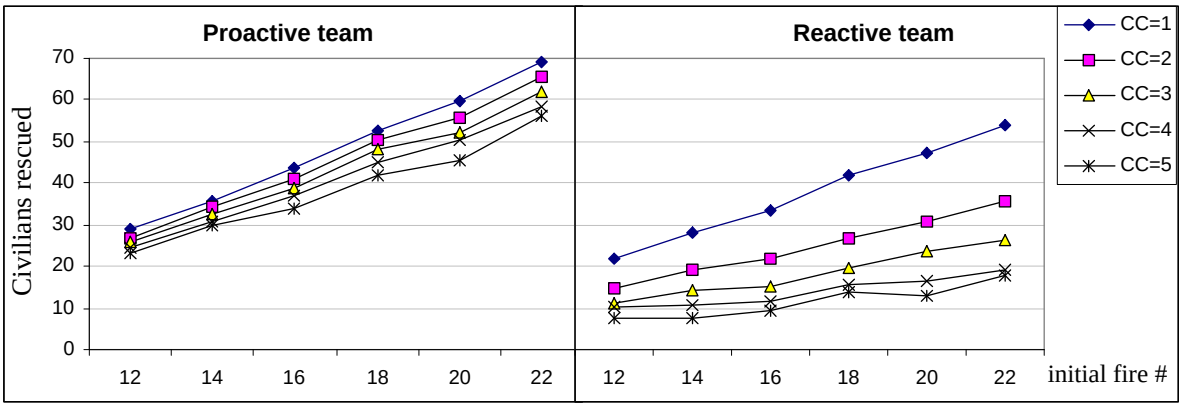

Fig. 7. Experiment II

shifted from the information consumer to the information provider, who knows exactly when to provide information.

For the two proactive teams, P team outperformed S team, and their performance gap increased as team task (fire number) increased. This is because P team uses the context-centric approach to manage teammates' information-needs, hence the ambulance will not send out irrelevant information in phase II. There is only one context change occurred in this experiment. From this we can anticipate that team performance may benefit more from the new approach as team complexity increases (e.g., team process involves more context changes) or the amount of information to be exchanged expands.

C. Varying Communication Costs

To gain a better understanding of how communication cost may affect team performance in our experimental setting, we varied the delay caused by inter-agent communication from 1 to 5 processing cycles. In this experiments we only focused on the proactive team (P team) and the reactive team (R team), using the same deployment of the simulator. As listed in Table 1, their communication behavior differs only in phase I of their team process.

We run 50 experiments for each team with a fixed number of fires and a fixed communication delay. Figure 7 shows the average number of civilians rescued by the two teams.

The results show that the team performance (in terms of number of civilians rescued) of the proactive team is significantly better than the reactive team; this confirms our prediction

that proactive communication works better than reactive communication in time-stress domains. It's not surprising to see that both teams' performance degraded as the communication cost increased. However, the results also show that the reactive team is more sensitive to the change of communication cost. This is because agents in the reactive team, not knowing when is the best opportunity to ask, need to initiate more communications. On the other hand, such sensitivity becomes less obvious as the communication cost increases. This is because the increase of the communication cost virtually decreases the frequency of communication.

VI. Comparison and Discussion

Recently there have been many efforts using graphs to represent multi-agent cooperations and to support social reasoning. We compare our approach with the dependence structures (networks and graphs) [17], the TAEMS approach [27], and the hierarchical task networks [16] used in RETSINA [28].

External description is the basic concept of the dependence theory proposed by Sichman, Conte, et al. [17], [18]. The external description of an agent reflects the agent's private view of all the agents in the system; it has one entry for each agent, containing the goals, actions, resources and plans of that agent. Such information can be used to infer dependence relations and to construct dependence networks and dependence graphs. A dependence network (or dependence graph), which records information regarding the action/resource dependences among agents, can be used to identify the dependence situations (e.g., mutual dependence, reciprocal dependence, among-dependence, collective-dependence, etc.) that may hold among two or more agents wrt. certain goals.

As the authors [18] noted, the consequence of using the dependence networks is the decreasing of inter-agent communications. From this perspective, the dependence structures and the INGs proposed in this paper serve the same ends. However, even though both are mechanisms useful for social reasoning, they differ in several ways due to their different focuses and the underlying research motivations. First, the dependence framework focuses on action and resource dependences emerging in carrying on a certain plan. For this reason, the dependence framework does not pay much attention to the dynamics of dependence relations, assuming the models of others are fixed once established. The ING approach focuses more on anticipating the dynamic information dependence and how the agents proactive meet others' needs. The INGs thus support

such dynamic reasoning. Second, in the dependence framework, an agent distinguishes his own and others' plans so that he can clearly identify various dependence situations (i.e., locally or mutually). In CAST, however, team plans are shared. Consequently, all agents in a team have the same INGs, although the INGs may not be exactly in agreement with the progress status.

In spite of such differences, the dependence theory could shed light on our future work. For instance, the current CAST system emphasizes more on encouraging proactive communication, which necessitates the reasoning on "who depends on me". In this regard, CAST can benefit from the idea of dependence structures, which allows an agent to reason about other teammates in two different perspectives: whom do I depend on, and who depends on me [17]. Some kinds of dependence relations characterized in the dependence framework [18] are also captured in INGs. For instance, intuitively we can take the relation between an information needer and the set of potential providers as "OR-dependence".

TAEMS (Task Analysis, Environment Modeling, and Simulation) is a framework for modeling complex task environments at three levels of abstraction: objective, subjective, and generative [27]. A TAEMS task network is a hierarchical representation of agent problem solving processes that describe alternative ways of accomplishing a desired goal. They represent major tasks and decision points, resource constraints, interactions between tasks, and primitive actions statistically characterized in terms of quality, cost and duration [29]. Thus, TAEMS-based agents can reason about what they should do and when, reason about temporal and resource constraints, and reason about interactions between activities being carried out by different agents [30], [31]. Interdependency between agents (represented in TAEMS by links to non-local effects) is the drive behind coordination.

One similar thing between TAEMS agents and CAST agents is that both take task structures (TAEMS task networks, PrT nets) as a critical component of their SMM. Such SMM maintains critical process information for run-time coordination. CAST takes further advantages of the PrT net representation of teamwork processes by generating INGs to facilitate the reasoning of others' information needs that may change over time. TAEMS task structures and INGs have different annotation information due to their different objectives. The primitive actions in a TAEMS task structure are associated with constraints about the quality, cost and duration, which serves in deciding potential coordination opportunities. The P-nodes and C-nodes are associated with information needs structured as inference trees, which allow agents to reason about teammates'

direct or indirect needs at multiple levels.

RETSINA-MAS [28] is a model of teamwork built on top of the RETSINA individual agent architecture [32]. In RETSINA-MAS system, initially all agents have a commonly agreed partial plan for fulfilling a team task (goal). The team members cannot commit to executing the team plan until after they have reached a consensus that all plan requirements are covered by their role proposals without any conflicts. A team plan is structured in Hierarchical Task Network (HTN) [16], where leaf nodes represent primitive actions and non-leaf nodes represent intermediate goals or compound tasks that must be achieved before the overall goal (root node) is complete.

Both CAST (Yen, et al., 2001) and RETSINA-MAS assume that all agents begin with their own copy of partially instantiated shared plans (recipes): CAST agents share the same team processes represented by PrT nets, while RETSINA-MAS agents share the same description of the task in the form of an HTN plan. In addition, both architectures endeavored to allow agents to monitor the progress when executing a team plan. In RETSINA-MAS the notion of "checkpoint" is used as a way of verifying the progress in relation to the overall team plan. Checkpoints are communicated by teammates to indicate individual progress. While, in CAST the communication of teamwork progress (control token information) is implemented as a kind of built-in information-needs. CAST agents need to inform appropriate teammates when they start a team plan, a team operator, a joint-action, or when the execution of a plan is terminated. From a more general point of view, the (pre-, termination, or preference) conditions or constraints evaluation along a team process in CAST can be taken as implicit checkpoints.

Various notions of context can be found in the field of artificial intelligence. For example, the notion of context is used by Giunchiglia [33] as a means of formalizing the idea of localization. Contexts in the SharedPlans theory [23], [12] serve as constraints or explanations of the adoption of intentions. Different from these theoretical studies, the notion of context in this paper is used to facilitate the practical reasoning and flexible management of teammates' time-changing information needs.

VII. Conclusion

CAST is a team-oriented agent architecture that explicitly represents a shared mental model (SMM) about team processes such that agents in a team can anticipate information needs of teammates and to proactively offer relevant information to them based on the SMM.

To overcome the limitations of the IFT-based management of teammates' information needs, in this paper, we described an ING-based mechanism for tracking the contextual changes of information needs. This enables agents to activate and deactivate information needs dynamically as a team activity proceeds. This approach, leveraging the shared awareness of teamwork processes, offers a practical way for agents in a team to exchange information proactively, selectively, and context-sensitively.

By experiments we evaluated the ING-based approach. The results demonstrated that the context-centric management of information needs can enhance team performance considerably. This is by no means conclusive, though. One of our future work is to further evaluate this approach in various domains with increasing complexities to explore the trend of its impacts on the performance of mixed human/agent teams.

ACKNOWLEDGMENTS

This research has been supported by AFOSR MURI grant No. F49620-00-1-0326.

REFERENCES

[1] J. Grunig, "Information and decision making in economic development," *Journalism Quarterly*, pp. 565–575, 1969.

[2] J. Larson and C. Christensen, "Groups a problem-solving units: Toward a new meaning of social cognition," *British Journal of Social Psychology*, pp. 5–30, 1993.

[3] J. Crant, "Proactive behavior in organization," *Journal of management*, vol. 26, no. 3, pp. 435–462, 2000.

[4] T. Dickinson and R. McIntyre, "A conceptual framework for teamwork measurement," in *Team performance assessment and measurement: theory, methods and applications*, M. T. Brannick, E. Salas, and C. Prince, Eds., 1997, pp. 19–44.

[5] R. McIntyre and E. Salas, "Measuring and managing for team performance: emerging principles from complex environments," in *Team effectiveness and decision making in organizations*, R. Guzzo and E. Salas, Eds. San Francisco: Jossey-Bass, 1995, pp. 149–203.

[6] K. A. Smith-Jentsch, J. H. Johnson, and S. C. Payne, "Measuring team-related expertise in complex environments," in *Making Decisions Under Stress: Implications for Individual and Team Training*, J. A. Cannon-Bowers and E. Salas, Eds., 2000, pp. 61–87.

[7] J. Yen, J. Yin, T. Ioerger, M. Miller, D. Xu, and R. Volz, "CAST: Collaborative agents for simulating teamworks," in *Proceedings of IJCAI'2001*, 2001, pp. 1135–1142.

[8] P. Stone and M. Veloso, "Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork," *AI*, vol. 110, pp. 241–273, 1999.

[9] B. J. Clement and A. C. Barrett, "Continual coordination through shared activities," in *Proceedings of the Second International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'03)*, 2003.

[10] M. Tambe, "Towards flexible teamwork," *Journal of Artificial Intelligence Research*, vol. 7, pp. 83–124, 1997.

[11] P. R. Cohen, H. J. Levesque, and I. A. Smith, "On team formation," *Contemporary Action Theory*, 1997.

[12] B. Grosz and S. Kraus, "The evolution of sharedplans," *Found. and Theories of Rational Agencies*, pp. 227–262, 1999.

[13] J. A. Cannon-Bowers, E. Salas, and S. A. Converse, "Shared mental models in expert team decision making," in *Individual and group decision making*, 1993, pp. 221–246.

[14] J. Yen, X. Fan, and R. A. Volz, "Information needs in agent teamwork," *Web Intelligence and Agent Systems: An International Journal*, no. 4, pp. 231–247, 2004.

[15] X. Fan, R. Wang, B. Sun, S. Sun, and J. Yen, "Multi-Agent Information Dependence," in *Proceedings of the 2005 IEEE International conference on Integration of Knowledge Intensive Multi-Agent Systems*, 2005, pp. 41–46.

[16] M. Williamson, K. S. Decker, and K. Sycara, "Unified information and control flow in hierarchical task networks," in *Proceedings of the AAAI-96 workshop on theories of planning, action, and control*, 1996.

[17] J. S. Sichman, R. Conte, C. Castelfranchi, and Y. Demazeau, "Social Reasoning Mechanism Based on Dependence Networks," in *Proceedings of the 11th European Conference on Artificial Intelligence*, A. G. Cohn, Ed., 1994.

[18] J. S. Sichman and R. Conte, "Multi-agent dependence by dependence graphs," in *AAMAS 2002*, 2002, pp. 483–490.

[19] 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 *Proc. of the 4th International Conference on Autonomous Agents*, 2000, pp. 427–434.

[20] D. Alberts, J. Garstka, and S. F., "Network centric warfare, ccrp."

[21] J. Yen, X. Fan, S. Sun, T. Hanratty, and J. Dumer, "Agents with shared mental models for enhancing team decision-makings," *Decision Support Systems*, p. (in press), 2005.

[22] X. Fan, J. Yen, M. Miller, and R. A. Volz, "The semantics of MALLET–an agent teamwork encoding language," in *Declarative Agent Languages and Technologies II: Second International Workshop, DALT 2004*, J. Leite, A. Omicini, and P. Torroni, Eds. Springer-Verlag, Berlin, 2005, vol. 3476, pp. 69–91.

[23] B. Grosz and S. Kraus, "Collaborative plans for complex group actions," *Artificial Intelligence*, vol. 86, pp. 269–358, 1996.

[24] E. Davis, "Knowledge preconditions for plans," *Journal of Logic and Computation*, vol. 4, no. 5, pp. 721–766, 1994.

[25] P. R. Cohen and H. J. Levesque, "Teamwork," *Nous*, vol. 25, no. 4, pp. 487–512, 1991.

[26] K. E. Korf, "Depth-first iterative deepening: an optimal admissible tree search," *Artificial Intelligence*, vol. 27, no. 1, pp. 97–109, 1985.

[27] K. Decker, "TAEMS: A framework for environment centered analysis & design of coordination mechanisms," in *Foundations of Distributed Artificial Intelligence, Chapter 16*. G. O'Hare and N. Jennings (eds.), Wiley Inter-Science, January 1996, pp. 429–448.

[28] J. Giampapa and K. Sycara, "Team-oriented agent coordination in the RETSINA multi-agent system," in *tech. report CMU-RI-TR-02-34, Robotics Institute, Carnegie Mellon University*, 2002.

[29] K. S. Decker and V. R. Lesser, "Quantitative modeling of complex computational task environments," in *Proceedings of the 11th National Conference on Artificial Intelligence*, 1993.

[30] T. Wagner, V. Guralnik, and J. Phelps, "A key-based coordination algorithm for dynamic readiness and repair service coordination," in *Proceedings of the 2nd International Conference on Autonomous Agents and MAS*, 2003.

[31] J. Graham, K. Decker, and M. Mersic, "DECAF - a flexible multi agent system architecture," *Autonomous Agents and Multi-Agent Systems*, vol. 7, no. 1–2, pp. 7–27, 2003.

[32] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng, "Distributed intelligent agents," *IEEE Expert, Intelligent Systems and their Applications*, vol. 11, no. 6, pp. 36–45, 1996.

[33] F. Giunchiglia, "Contextual reasoning," *Epistemologia, special issue on I Linguaggi e le Macchine*, vol. XVI, pp. 345–364, 1993.