Monitoring Deployed Agent Teams

Gal A. Kaminka
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213
galk@cs.cmu.edu

David V. Pynadath
Information Sciences Institute
Univ. of Southern California
Marina del Rey, CA 90292
pynadath@isi.edu

Milind Tambe
Information Sciences Institute
Univ. of Southern California
Marina del Rey, CA 90292
tambe@isi.edu

ABSTRACT

Recent years are seeing an increasing need for on-line monitoring of deployed distributed teams of cooperating agents, e.g., for visualization, or performance tracking. However, in deployed systems, we often cannot rely on the agents to communicate their state to the monitoring system: (a) we rarely can change the behavior of already-deployed agents to communicate the required information (e.g., in legacy or proprietary systems); (b) different monitoring goals require different information to be communicated (e.g., agents' beliefs vs. plans); and (c) communications may be expensive, unreliable, or insecure. This paper presents a non-intrusive approach based on plan-recognition, in which the monitored agents' state is inferred from observations of their routine actions. In particular, we focus on inference of the team state based on its observed *routine* communications, exchanged as part of coordinated task execution. The paper includes the following key novel contributions: (i) a *linear time* probabilistic plan-recognition algorithm, well-suited for processing communications as observations; (ii) an approach to exploiting general knowledge of teamwork to predict agent responses during normal execution, to reduce monitoring uncertainty; and (iii) a monitoring algorithm that trades expressivity for scalability, representing only certain useful monitoring hypotheses, but allowing for any number of agents and their different activities, to be represented in a single coherent entity. Our empirical evaluation illustrates that monitoring based on observed routine communications enables significant monitoring accuracy, while not being intrusive. The results also demonstrate a key lesson: A combination of complementary low-quality techniques is cheaper, and better, than a single, highly-optimized monitoring approach.

1. INTRODUCTION

Recent years have seen tremendous growth of fully-deployed applications involving distributed, heterogeneous, multi-agent teams (e.g., [6, 16]). This growth has lead to increasing need for monitoring techniques that allow a synthetic agent or human operator to monitor and identify the state of the distributed team. Such monitoring is critical for visualization [13], for identifying failures in execution [6, 11], and for facilitating collaboration between the monitoring agent and the members of the team [4].

This paper focuses on techniques that a monitoring agent may use to accurately identify the state of a distributed team, for instance to aid a human operator in tracking the team's progress. We seek practical techniques that are appropriate for realistic, large-scale applications, and in particular techniques that are appropriate for monitoring previously-deployed applications.

One approach to monitoring expects each monitored team-member to communicate its state to the monitoring agent at regular intervals, or at least whenever the team-member changes its state. This provides the monitoring agent with accurate information on the state of the team. Unfortunately, this monitoring approach (henceforth, *report-based monitoring*) suffers from several problems in monitoring deployed distributed teams:

1. It involves intrusive modifications to the behavior of monitored agents, to cause them to communicate their state. However, many deployed distributed systems are built from heterogeneous agents, developed by different organizations at different times. This makes modifications difficult. In particular, legacy and proprietary systems are notoriously difficult to modify.

2. The information which the agents must communicate may change depending on the monitoring task (e.g., visualization vs. fault-detection). Requiring agents to change their reports (based on monitoring tasks which may have not been foreseen) undermines the scalability of this approach.

3. It places heavy computational and bandwidth burdens on the monitored agents and the communication lines involved, *as has been repeatedly noted in the literature* (e.g., [9, 10, 4, 5, 16, 21]). In our own application domain, the agents normally exchange approximately 100 messages during task execution. However, using reports, they will need to communicate 50,000 messages. This difficulty only increases with a scale-up in the number of agents.

4. It assumes completely reliable and secure communications between the team-members and the monitoring agent. Unfortunately, this is often not possible. Failures do occur, and communications must often be limited in adversarial settings for security reasons.

An alternative monitoring approach is based on plan-recognition (e.g., [17, 8, 11]): The monitoring agent infers the unobservable state of the agents based on their observable actions, using knowledge of the plans that give rise to the actions. This approach is

AGENTS'01 May 28-June 1, 2001, Montréal, Québec, Canada.

completely non-intrusive, requiring no changes to agents' behaviors, and allows for changes in the requested monitoring information. Furthermore, plan-recognition can rely on inference to compensate for occasional communication losses, something not possible in report-based monitoring, and is thus much more robust to communication failures.

Generally, the only observable actions of agents in a distributed team are their *routine* communications, which the agents exchange as part of task execution [13]. Unfortunately, most local actions taken by an agent (for instance, displaying information on a console) are not observable to a remote monitoring agent (that may be running hundreds of miles away). Fortunately, the growing popularity of agent integration tools [20, 14] and agent communication languages [3] increases standardization of aspects of agent communications, and provides increasing opportunities for observing and interpreting inter-agent communications.

Given knowledge about the plans the agents may be executing, a monitoring agent using plan-recognition can infer the agents' current state from such observed routine messages. However, uncertainty in plan-recognition is a well-recognized major difficulty. Such uncertainty (i) may mislead the monitoring agent in its decision making, and (ii) can be expensive to reason about, as the number of hypotheses grows exponentially in the number of agents.

Indeed, such is the case in observing communications. Agent team members cannot and do not in practice continuously communicate among themselves about their state [10, 4]. Furthermore, communications sometimes occur in small subteams (i.e., only a single agent or members of a particular subteam will communicate): In our application, the agents roughly communicate once for every twenty state-changes on average, with some agents only communicating once or twice during the entire task execution. And yet, the monitoring system must infer the state of all agents in the team, at all times.

This paper presents a number of novel techniques used by OVERSEER, a fully-implemented monitoring agent, to reduce the uncertainty and increase the efficiency of monitoring potentially large distributed teams, under the severe response-time and scale-up requirements imposed by realistic applications. OVERSEER is regularly used in monitoring complex, distributed applications composed of heterogeneous agents. It includes the following key contributions: (i) an efficient, *linear time* probabilistic plan-recognition mechanism, particularly well-suited for processing communications in agent teams; (ii) a method for exploiting the procedures used by a team to effectively predict (and hence effectively monitor) team responses during normal and failed execution; and (iii) YOYO*, an algorithm that models the agent team (with all the different parallel activities taken by individual agents) using a single structure, instead of modeling each agent individually—sacrificing some expressivity (the ability to accurately monitor the team in certain coordination failure states) for efficiency and scalability. In particular, YOYO* runs in linear time by restricting itself to coherent monitoring hypotheses, while a more general approach that avoids these restrictions runs the risk of run times exponential in the number of agents.

In applying these techniques, OVERSEER mainly utilizes two knowledge-bases to carry out its task: (a) knowledge of the *plan hierarchy* involved in carrying out the task (e.g., knowledge that a particular sequence of steps in the plan calls for a *route-planner agent* to plan routes and pass them to *helicopter agents*); and (b) the team's organizational hierarchy (*team-hierarchy*), which specifies which agents and subteams are part of what teams. The team-hierarchy enables OVERSEER to engage in *Socially-Attentive Monitoring* [11], exploiting knowledge about the relationships that the agents ideally maintain, and the procedures that they take to maintain these relationships. In addition, OVERSEER utilizes a computationally cheap, but low-accuracy, temporal model to reason about plan durations.

We present a rigorous evaluation of OVERSEER's monitoring capabilities in one of its application domains (described in Section 2) and show that the techniques presented result in a significant boost to OVERSEER's monitoring accuracy and efficiency, while sacrificing little in terms of being able to detect failures. While previous work in multi-agent plan recognition has either focused on exploiting explicit teamwork reasoning, e.g., [17], or explicitly reasoning about uncertainty, e.g., [8], a key novelty in OVERSEER is that it effectively blends these two threads together.

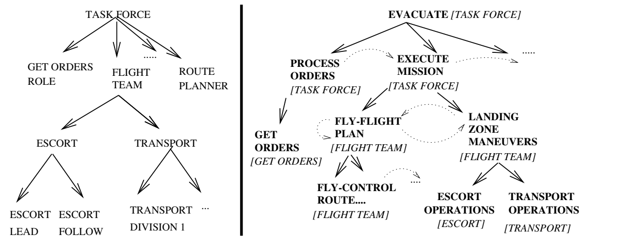
2. AN EXAMPLE DISTRIBUTED TEAM

OVERSEER has been applied in multiple applications, monitoring distributed teams of heterogeneous, software agents, consisting of 10 to 20 team members who are collaborating across the Internet. In this section, we describe one of these applications, which we have used to evaluate OVERSEER. In this application, a distributed team of 11 agents is executing a simulation of an evacuation of civilians from a threatened location. The integrated system allows a human commander to interactively provide locations of the stranded civilians, safe areas for evacuation and other key points. Simulated helicopters then fly a coordinated mission to evacuate the civilians, relying on various information agents to dynamically obtain information about enemy threats, (re)plan routes to avoid threats and obstacles, etc. The distributed team is composed of diverse agents from four different research groups: A Quickset multimodal command input agent [1], a route planner [15], the Ariadne information agent [12] and eight synthetic helicopter pilots [19].

The team is integrated using the Teamcore multi-agent integration architecture [20], which accomplishes integration by "wrapping" each agent with a proxy that maintains collaboration with other agents (via their own proxies). A distributed application is formed by a team of agents jointly executing the application task, described by a *team-oriented program*. This program consists of a set of hierarchical team plans, with assigned roles for teams and subteams. As an example, Figure 1-a shows a part of the team/subteam hierarchy used in the evacuation-domain (described below). Here, for instance, TRANSPORT is a subteam of Task-Force. Figure 1-b shows an abbreviated plan-hierarchy for the same domain. High-level team plans, such as EVACUATE, typically decompose into other team plans, such as PROCESS-ORDERS, and, ultimately, into leaf-level plans that are executed by individuals. Temporal transitions are used to constrain the order of execution of plans. There are teams assigned to execute the plans, e.g., *Task Force* team jointly executes EVACUATE, while only the TRANSPORT subteam executes the TRANSPORT-OPS step. The team-oriented program for this application consists of about 40 team-plans.

To execute the team-oriented program, each proxy uses a domain-independent teamwork model, called STEAM [18]. STEAM allows the agents to automatically coordinate using selective communications, exchanging messages which either attempt initiation of a joint-execution of a team-plan, or termination of such a joint-execution (similarly to normal teamwork protocols). Normally, agents only communicate about some plans, and only rarely do they announce both initiation and termination of a specific plan. Also, termination messages are usually sent only by a single team-member, the first to discover a cause for termination of joint execution.

OVERSEER monitors these routine communications, and allows



(a) (b)

Figure 1: Portions of the team-hierarchy (a) and plan-hierarchy (b) used in our domain. Dotted line show temporal transitions.

humans and agents to query about the present and future likely plans of the entire team, its subteams and individuals—to monitor progress, compute likelihoods of failure, etc.

Several key concerns have led us to rely on plan-recognition in building OVERSEER. The main concern has been the difficulty of modifying agent behavior to support report-based monitoring. There are several difficulties here: First, the agents are already deployed in several places, e.g. government laboratories and universities. Modifying the agents at each deployed location is problematic. Second, such modifications are intrusive—they interfere with carefully designed timing specifications of given tasks, requiring further modifications by other agents developers. Third, the distributed nature of TEAMCORE implies that there is no centralized server which controls the behavior of the agents, but instead changes are required in the different proxy types. Fourth, the evacuation task is but one of different applications using TEAMCORE (e.g., see [20]), and thus arbitrary modifications in one application would require consideration of other applications. Indeed, as monitoring requirements change, the information needed about each agent changes as well. A solution that would require constant redesign of the agents and architecture such that they report to accommodate changing monitoring needs will not scale up.

A second concern has been the prohibitive communication costs of report-based monitoring. In a team of 11 (used as an example in this paper), regularly scheduled state reports from the agents at the required temporal resolution would require approximately 50,000 messages to be sent during a 15-minute run, with the number nearly doubling when we reach 20 agents. If we instead have the 11 agents only report on state changes, announcing plan initiation and termination, approximately 2,000 messages have to be sent. However, this is still an order-of-magnitude more than the normal 100 messages or so that are exchanged by the 11 agents as part of routine execution.

We emphasize that we do not rule out the use of report-based monitoring using communications. Our choice of a plan-recognition approach stems from the problems we have found in using reports with an already-deployed application. When it is possible, focused communications in service of monitoring can ease the monitoring task [20].

3. INDIVIDUAL-BASED MONITORING

This section introduces a novel plan-recognition algorithm which underlies OVERSEER's monitoring of each agent in a team, individually. The algorithm is particularly suited for monitoring communications, and allows for linear-time inference. It uses an approach to monitoring each single agent, based on observed communications originated by the agent, and on predictions of plan-execution duration. To monitor multiple agents, our initial attempt in OVERSEER was to build an array of such single-agent plan-recognizers, one for each agent. OVERSEER then maintains the state of the agents by updating their plan-hierarchy based on messages that are observed, and the time that has passed.

For instance, if OVERSEER observes a message about the initiation of FLY-FLIGHT-PLAN by one of the helicopters, then it knows from Figure 1b that PROCESS-ORDERS cannot be a possible future plan of the agent. However, there is uncertainty as to whether FLY-FLIGHT-PLAN and LANDING-ZONE-MANEUVERS are active, as both are possible future states, and the duration of FLY-FLIGHT-PLAN is not certain. Furthermore, since OVERSEER does not observe any message hinting at the state of other team-members, it updates their hypothesized state based on estimated plan-duration , taking into account the time that has passed since the last update.

We address the uncertainty in such monitoring through a probabilistic model that supports quantitative evaluation of the hypotheses. Since we monitor agents in terms of the selected plans, we use a time series of state variables, where, at each point of time, the agent's state is the state of the team-oriented program that it is currently executing, i.e., a path from root to leaf in the team-oriented program tree. We represent the plans in the program by a set of boolean random variables, $\{X_{i,t}\}$, where each variable $X_{i,t}$ is true if and only if plan X_i is active at time t. Beliefs about the agent's actual state at time t are then represented as a probability distribution over all variables $\{X_{i,t}\}$.

In the example above, OVERSEER would assign (at time t, after observing the message) a probability of 0 to PROCESS-ORDERS ($b_t(ProcessOrders) = 0$), and a probability of 0.5 to each of FLY-FLIGHT-PLAN and LANDING-ZONE-MANEUVERS (assuming it can rule out the plans that follow EXECUTE-MISSION)— $b_t(FlyFlightPlan) = b_t(LandingZoneManeuvers) = 0.5$.

OVERSEER begins with a belief that the agent is executing its top-level plan at time 0 (with certainty, i.e., $b_t(Evacuate) = 1.0$). If it observes a message from an agent, it incorporates the evidence into its beliefs about the sender, according to the method described in Section 3.2. If it does not observe a message from an agent, it propagates belief about currently executing plans throughout the hierarchy representing this agent, using the method described in Section 3.1 to simulate plan execution in time.

3.1 Belief Update with No Observation

If OVERSEER does not observe communication, then it rolls the model forward to the next time slice. For each plan that the agent could be executing, it computes, using a plan-duration model (*temporal model*) how likely it is that the agent will complete execution and go on to its next plan. For simplicity, we treat the duration of a leaf plan, X, as an exponential random variable, where the probability of the plan lasting more than τ time units decays exponentially as $e^{-\tau\lambda_X}$. The parameter λ_X then corresponds to 1/(average duration of X), and can be acquired from domain experts or previous runs. Given this model of plan duration, the probability of the plan's completion between times t and $t+1$ is $\Pr(done(X,t)|X_t) = 1 - e^{-\lambda_X}$.

Once it uses the temporal model to compute the probability of plan termination, OVERSEER determines which plan the agent will execute next. It examines the possible successors and computes the probability of taking the corresponding transition, conditioned on the fact that no message was observed. For each plan X, OVERSEER uses the probability of entering each successor, Y, given that X has just completed: $\pi_{xy} = \Pr(Y_{t+1}|X_t, done(X,t))$. It also uses the probability of seeing a message given the transition,

$\mu_{xy} = \Pr(msg_t|X_t, Y_{t+1})$. The parameters μ, and π, can be acquired from previous runs as well.

OVERSEER makes a Markovian assumption that the plan history before time t does not affect the probabilities. It then combines these values to get the desired conditional probability:

$$\begin{aligned} & \Pr(Y_{t+1}|X_t, done(X,t), \neg msg_t) \\ = & \frac{(1-\mu_{xy})\pi_{xy}}{\Pr(\neg msg_t|X_t, done(X,t))} \\ = & \frac{(1-\mu_{xy})\pi_{xy}}{\eta_X} \end{aligned}$$

The normalizing denominator, η_X, is the sum of the numerator over all possible successors, Y, and can be pre-computed off-line. If *all* possible transitions *require* a message, then η_X will be zero. In this case, the agent cannot have begun execution of any successor, even though it has completed execution of X. We use a *blocked* state associated with each plan to indicate this contingency. Thus, $b_t(X, \neg block)$ is our belief at time t that X is executed by the monitored agent, and has not terminated; $b_t(X, block)$ is our belief that X has terminated, but the agent has not begun execution of a successor.

If a particular transition indicates the termination of the entire execution path, then the probability of the transition corresponds to the probability that the *parent* plan has completed. Intuitively, the probability that we have terminated a last child is the probability that we have terminated execution of the parent. We therefore compute the probability of transitions out of the parent plan using the last child's completion probability.

If the plan has children, then we must also distribute the incoming probability (from parent or previous plans) among them (PROPAGATE-DOWN). Since we assume that all plans take at least a single time step to complete, we consider only the first child when we first propagate probabilities from the parent. In Figure 1b, upon first entering the top-level plan EVACUATE, the only possible child plan that can be active at time 0 is PROCESS-ORDERS. If there are multiple first children then they denote alternative execution paths for a single agent, and we compute the probability over them by dividing the probability incoming to the parent among them. If any children have child plans of their own, this new incoming probability is distributed in turn, using the same method. In the next timestep, we propagate not only from the parent to its first children, but also from these children to the next child in order of execution, etc. Algorithm 1 presents the pseudo-code for the overall propagation computations, calling the PROPAGATE-DOWN function for this downward update.

Algorithm 1 PROPAGATE-FORWARD(**beliefs** b, **plans** M)

for all plans $X \in M$ **do**
  $b_{t+1}(X, \neg block) += b_t(X, \neg block)e^{-\lambda_X}$
  if $\eta_X = 0$ **then** {Message required}
    $b_{t+1}(X, block) \leftarrow b_t(X, \neg block)(1 - e^{-\lambda_x}) + b_t(X, block)$
  else {Message not required}
    for all plans Y that succeed X **do**
      $\rho \leftarrow b_t(X, \neg block)(1 - e^{-\lambda_x})(1 - \mu_{xy})\pi_{xy}/\eta_x$
      if $Y = done$ **then**
        $b_{t+1}(parent(X), block) += \rho$
      else {Y is a sibling plan}
        $b_{t+1}(Y, \neg block) += \rho$
        PROPAGATE-DOWN(Y, ρ, b, M)

3.2 Belief Update with Observed Message

While observing team communication, we can expect to see messages sent by an individual member that identify either plan initiation or termination. Suppose we have observed a message, *msg*, that corresponds to initiation. Then, if only one plan, X, is consistent with *msg*, then we know, with certainty, that the agent is executing X, regardless of whatever evidence we have previously observed, i.e., $\Pr(X_t|msg_t, evid_{t-1}) = 1$. If multiple plans are consistent with *msg*, we distribute the unit probability over each plan, weighted by any prior belief in seeing the given message.

If we observe a message indicating the termination of X, then we know that the agent was executing X in the previous time step but that it has moved on to some successor. Thus, for each state, Y, that can follow X, we set our belief of Y to be proportional to a transition probability, similar to those in Section 3.1, except that we are now conditioning on observing a message. Algorithm 2 presents the pseudo-code for the complete procedure for incorporating observational evidence.

Algorithm 2 INCORPORATE-EVIDENCE(**msg** m, **beliefs** b, **plans** M)

for all plans $X \in M$ consistent with m **do**
  if m is an initiation message **then**
    $b'(X, \neg block) \leftarrow b_t(X, \neg block)$
  else {m is a termination message}
    for all plans $Y \in M$ that succeed X **do**
      $b'(Y, \neg block) \leftarrow b_t(X, block)\mu_{xy}\pi_{xy}/(1 - \eta_x)$
normalize distribution b'
for all plans $X \in M$ with $b' > 0$ **do**
  $b_{t+1}(X, \neg block) \leftarrow b'(X, \neg block)$
  $b_{t+1}(parent(X), \neg block) += b'(X, \neg block)$
  PROPAGATE-DOWN($X, b'(X, \neg block), b, M$)

3.3 Individual Agent Recognition Complexity

The pseudo-code of Algorithms 1–2 demonstrates that both types of belief updates have a time complexity linear in the number of plans and transitions in M. We gain this efficiency from two sources. First, the Markovian assumption in the temporal model allows our propagation algorithm to reason forward to time $t + 1$ based on only our beliefs at time t, without regard for previous history. Second, we make another Markovian assumption that the probability of observing a message depends only on a relevant plan being active and is independent of the past history. With that assumption, we can incorporate evidence, again, based on only our beliefs at time t.

4. EXPLOITING TEAMWORK

The previous section has outlined OVERSEER's basic approach, which runs plan-recognition separately for each agent, without considering the effects of an agent's communication on its peers. Using an array of individual models that are updated as messages come in, and with the passage of time, the state of a team is taken to be the combination of the most likely state of each agent, i.e., the approach takes a team to be a collection of individual agents.

This approach unfortunately proved insufficient by itself (see results in Section 5). The scarcity of observations makes monitoring more reliant on the temporal knowledge. But as there is high variance in the temporal behavior of the plans, the temporal knowledge was only of limited use. One way to tackle this difficulty would be to improve the temporal model, at the cost of losing its appealing computational efficiency and weak dependence on domain knowledge.

However, a key source of knowledge is not being used in the previous approach: A team is more than a collection of individuals: It involves social structures and procedures that maintain them [9], which can be used to better predict the behavior of team-members.

In a team, relationships and interactions among agents are predictable (at least to a degree), and this property can be used to alleviate uncertainty.

Such exploitation of social knowledge for monitoring is called *Socially-Attentive monitoring* [11]. The following sections report on several novel socially-attentive techniques that significantly improve monitoring accuracy and scalability. The techniques integrate team and teamwork knowledge as the basis for disambiguating among the monitoring hypotheses, eliminating hypotheses which are unlikely to be useful. Such elimination not only significantly boosts accuracy, but can also reduce the space requirements of monitoring.

4.1 Efficient Reasoning with Team Coherence

A key difference between monitoring a team and monitoring a group of individuals is that we expect the team to work *together:* Team-member are ideally in agreement about their joint goals and plans. This phenomenon—called *team-coherence* [11]—holds at different levels in the team. Agents in an atomic subteam work together on the plans selected for the subteam, subteams work together with sibling subteams on higher level joint plans, etc. Individual agents may still choose their own execution, but they do so in service of agreed-upon joint plans.

Assuming non-failure conditions, we can use coherence as a heuristic, preferring hypotheses in which team-members are in agreement about their joint plans, over hypotheses in which they are in disagreement. This heuristic requires knowledge about what plans in the hierarchy are to be executed by what (sub)teams (as encoded in the plan-hierarchy, part of the team-oriented program), and knowledge about what subteam/agent is part of another subteam (the team-hierarchy).

For example, suppose that the entire team is known to be executing FLY-FLIGHT-PLAN (Figure 1-b). Now, a message from one member of the TRANSPORT subteam is observed, indicating that it has begun execution of the TRANSPORT-OPS plan step. Since this plan step is to be jointly executed by all members of the TRANSPORT subteam (and only them), we can use coherence to prefer the hypothesis that the other subteam members have also initiated execution of TRANSPORT-OPS. Furthermore, since this plan-step is in service of the LANDING-ZONE-MANEUVERS plan, which is to be jointly executed by the TRANSPORT and ESCORT subteams, we can prefer the coherent hypothesis that team-members of ESCORT are executing LANDING-ZONE-MANEUVERS. We can now come back down the plan-hierarchy and infer that members of the ESCORT subteam are executing ESCORT-OPS, etc.

Preference for coherent hypotheses not only significantly promotes accuracy (see Section 5), but also offers hope for resolving scalability issues. Indeed, coherence is a very strong constraint, since there is in general only a linear number (in the size of the plan-hierarchy) of coherent hypotheses, but an exponential number of incoherent hypotheses. We can exploit this property of coherence in designing monitoring algorithms that reason only about coherent hypotheses, and therefore offer better scalability as the number of agents increases. Such algorithms may not be able to reason about incoherent hypotheses, and are therefore less expressive. However, the results demonstrate (Section 5) that the level of accuracy even with such limited expressiveness is more than sufficient for our purposes.

We present here the YOYO* algorithm, an efficient technique for reasoning about coherent hypotheses (Algorithm 3, building on Algorithms 1 & 2). Per its name, YOYO* climbs up and down the team- and plan-hierarchies (as in the example above), to realign prior knowledge to be coherent with the new observation. YOYO*'s key novelty is that it relies on a *single* plan-hierarchy to represent all team-members together, instead of an array of plan-hierarchies. It thus offers a much more scalable solution to monitoring the teams—space complexity remains constant as the number of agents increases (provided no changes are made to the plan-library). On the other hand, YOYO* sacrifices expressivity, as it cannot represent incoherent hypotheses (though it is able to detect their occurrence). For example, YOYO* cannot represent a hypothesis that one agent is executing LANDING-ZONE-MANEUVERS while the other one is executing FLY-FLIGHT-PLAN. However, YOYO* may be able to detect that a failure of this type occurred.

Algorithm 3 YOYO*(**plan-hierarchy** M, **team-hierarchy** H)

```
1:  Loop forever:
2:  if messages rec'd–new plan S team T, then
3:      Incorporate-Evidence(T, S)
4:      tmp ← T
5:      while tmp is not the root team in H do
6:          find in M lowest common ancestor A of S joint to tmp
            and its sibling teams
7:          for each child transition of A whose subteam≠tmp do
8:              SCALE(the subtree roots at the child), so state probabil-
                ities of each subteam's child plan sums up to the new
                probability of A
9:          tmp ← parentof(tmp)
10: else
11:     PROPAGATE-FORWARD in M
```

The key idea in YOYO* is that once a step is taken upwards in the plan-hierarchy (line 6), it is followed by a traversal of the subtrees below the new root node such that all the evidence below the node is made coherent. This is done by the SCALE procedure, which re-distributes the new state probability of a parent among its children, such that each child gets scaled based on its relative weight in the parent. The end result is that the state probabilities of the children are made to sum up to the state probability of the parent. The process is recursive, but never re-visits a subtree.

YOYO* also requires minor modifications to PROPAGATE-FORWARD (Algorithm 1) and INCORPORATE-EVIDENCE (Algorithm 2). INCORPORATE-EVIDENCE must take a team T into account when incorporating evidence: Only transitions that T is allowed to take may be followed. PROPAGATE-FORWARD must address teams as well: Given some total outgoing probability (either to a sibling or child transition), if the outgoing transitions are to be taken by different teams (such as the TRANSPORT and ESCORT teams), the same total probability would be used for each transition, instead of splitting the outgoing probability between the transitions.

The following demonstrates how YOYO* handles the example given above (see Section 5 for in-depth evaluation): When the message from the member of the TRANSPORT subteam is observed, the new evidence is first incorporated for the transport team. Among other changes, the probability of the plan TRANSPORT-OPS goes up significantly. Then, YOYO* begins climbing up and down the team- and plan-hierarchies: It first finds the lowest common ancestor of TRANSPORT-OPS that is shared by the TRANSPORT team and its sibling. This is the LANDING-ZONE-MANEUVERS plan. It has one child that is to be taken by the ESCORT team (different than TRANSPORT), and so the subtree pointed to by this child transition is scaled up–which means that the probabilities indicating that the ESCORT team is executing ESCORT-OPS goes up, based on evidence from the TRANSPORT team. The process then continues to EXECUTE-MISSION, etc.

4.2 Predicting Team Responses

Another key difference between monitoring a team as opposed to a group of individual agents, is that teams utilize procedures (sometimes called conventions [9]) by which they maintain their commitments. Knowledge of these procedures can be useful to alleviate uncertainty, since it allows us to make predictions as to the behavior of team-members in coordinating with one another. In particular, knowledge of the communication procedures used by the team allows predicting future observed messages. And knowledge of failure/recovery procedures allow predictions of failure states, given the current state of the team and the fault models the team uses in taking corrective actions.

The team's communication procedures determine at what points during the execution of the task the team will communicate, either establishing or terminating a plan. In many cases, a correct prediction that a transition will not be taken without a message being observed can completely rule out all but one hypothesis. For instance, if an establishment message is expected as agents take the transition from FLY-FLIGHT-PLAN to LANDING-ZONE-MANEUVERS (Figure 1), then the plan-recognition system, while not observing such a message, can eliminate (or at least rank much lower) the possibility that the team is executing LANDING-ZONE-MANEUVERS.

The knowledge required for such team-responses predictions can be acquired by learning. Indeed, simple rote-learning was sufficient in many cases to build a very effective model of the communication policy of an team. The model proved sufficiently effective to allow OVERSEER to determine that a particular message was delayed or missing, thus detecting failures that we had not considered originally. However, the learned predictions did not fare well in handling rare communication patterns, such as those occurring during certain rare failures.

5. RESULTS AND EVALUATION

OVERSEER is a fully implemented system, which is used in actual runs of the evacuation application over the Internet. The first part of the evaluation tests the contribution of the different techniques in OVERSEER to recognizing the correct state of the agents. Figure 2 compares the average accuracy for a sample of our actual runs, marked 'A' through 'J' (X-axis). In each run, the team executed its task completely. At different points during the 10–20 minute execution, the *actual* state of the system was compared to the state *predicted* by OVERSEER, where the prediction was taken to be the current most-likely hypothesis. Each run had 22–45 such comparisons (data-points). The average accuracy for each run across those comparisons is given in the 0-1 (0-100%) range, on the Y-axis. The results were found to be statistically significant with confidence greater than 99.999%.

The average accuracy when using the individual models with no coherence (as in Section 3) is presented in the leftmost bar (marked *Temporal*) in each group (Figure 2), and is clearly very low. This approach is a straightforward attempt at monitoring multiple agents by monitoring each individual, without considering the interactions between them. The next bar presents the average recognition accuracy when only coherence is used to rule out hypotheses (Section 4.1), with ties broken randomly. Coherence is clearly an effective constraint, as it brings the accuracy up by approximately 15% *without using any temporal knowledge*. The next bar to the right (*Coherent*, *Temporal*) presents the results of combining both coherence and the probabilistic temporal model (Sections 3 and 4.1). This approach offers a significant boost in accuracy, when compared to the approach relying on either information alone.

The next bar to the right (*Coherent, Comm*) shows the effects of

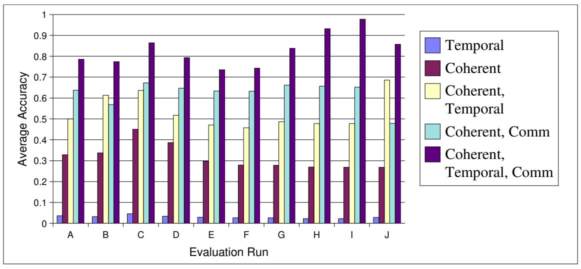


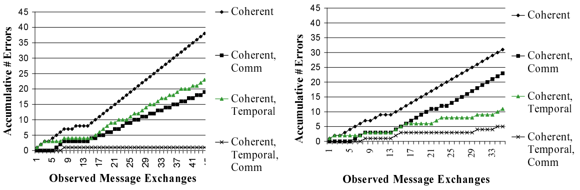
Figure 2: Average accuracy in sample runs.

combining the use of coherence with the use of predictions based on knowledge of the communication procedures used by the team (Section 4.2). Here, the communication predictions were used to restrict the set of coherent hypotheses considered, with ties broken randomly. The results show that while a temporal model can improve the accuracy of the recognition, alternatively, a significant boost in accuracy is achieved through the ability to predict at what point during task execution communication will occur. Indeed, in many cases, such predictions were able to rule out all hypotheses but one or two. The remaining bar (*Coherence, Temporal, Comm*) presents the average accuracy in each run using the combination of all techniques. Clearly, this combined technique is superior to the use of any single technique alone, or a simpler combination, resulting in average accuracy (across all runs) of 84%. While such high-accuracy is sufficient for our application, further improvements could be realized by improving the learned communication model. Indeed, OVERSEER's combination of techniques highlights a key lesson: Straightforward modeling of a team as a collection of individuals is insufficient, as it does not take the structured interactions between team-members into account.

To show the system in action, rather than summarized average accuracy, Figure 3 shows the system's results in two particular runs (I, J). The figure presents the accumulative number of errors as time goes by during the run, where an error means that the most likely hypothesis does not reflect the true state of the agent/team. Each message exchange corresponds to one to a dozen messages communicated by the agents, initiating or terminating a plan.

The line marked *Coherent* shows the cumulative number of errors if only coherence is used to select the correct hypothesis. Most such choices turn out to be erroneous since a random choice is made among the competing hypotheses. The line marked *Coherent, Temporal* shows the results for the same runs using both coherence and the temporal model to choose the most likely hypothesis. This approach is much more successful in run J than in I, as evident by the slower rise in the number of cumulative errors.

The line marked *Coherent, Comm* shows the results using coherence and the response predictions. In run I, it was able to fully compensate for the lack of temporal knowledge, generating approximately the same number of errors as resulted in *Coherent, Temporal*. However, in run J, the *Coherent, Comm* technique did not fare as well. This is at least partially because of agent failures that have occurred this run, that rendered the team communications predictions useless, since the agents did not use their usual communications, but were communicating instead about the failures. This clearly shows a limitation of the simple learning approach we took, and we intend to address it in future work. Finally, the remaining line displays the results of using coherence, the temporal model, and the model predicting communications. In both runs, we see again that this combined technique fares significantly better than either technique by itself.



(a) Run I (b) Run J

Figure 3: Accumulative number of errors in runs I and J.

There are several general lessons that emerge. First, communication predictions can be very effectively used instead of a more sophisticated, more accurate, temporal model. Our own application's temporal behavior has a lot of variance, but rather than relying on sophisticated temporal model (that may be computationally expensive), the communication predictions facilitate high levels of monitoring accuracy. Indeed, an important second lesson is that successful monitoring agents can integrate independent low-accuracy techniques, rather than focus on a particular higher-accuracy monitoring technique (such as a more sophisticated temporal model).

A second part of evaluating OVERSEER examines a key tradeoff between the expressivity and efficiency involved in the plan-recognition techniques we have presented. From the accuracy discussion above, it is clear that coherence is a useful heuristic. YOYO* takes an extreme approach, strictly ruling out reasoning about incoherences. It is impossible for YOYO*, for instance, to represent an incoherence in which two team-members are in disagreement about the plan executed by the common team. An approach in which each individual is represented separately allows for such representation, and in this respect is more expressive. However, *YOYO* is still able to detect many incoherences*–it would fail to find the states referred to in the messages and announce failure.

On the other hand, YOYO* offers great computational advantages when compared to the individual representation approach. YOYO* requires the team-hierarchy, of course, but only a *single*, fully-expanded plan-hierarchy to represent the entire team. This hierarchy is a union of all the individual agent plan-hierarchies. In the worst case, every agent carries out a completely different plan. In this case, YOYO*'s space complexity will be as inefficient as the individual recognition approach, requiring as much space for the plan-library, plus the small space for the team-hierarchy. However, a more typical case is where agents share many plans and transitions, because they carry them out jointly. In this case, YOYO* offers significant savings, since the duplicated portions of the invidivual plan-hierarchies would be unified into a single copy. *In our application domains the space savings were about 91%*: In monitoring a team of 11 agents, 66 plan-nodes were used in YOYO*, 726 nodes were used in the individual models array. The array also required 66 additional nodes with each additional monitored agent (assuming no addition to the plan-hierarchy itself), while YOYO* required only one additional node (in the team-hierarchy).

YOYO* builds on a key insight—that team activity is more than a collection of individual activities. This insight is used to exploit information about the activities of one team-member, in hypothesizing about the activities of another team-member. The accuracy results demonstrate that this is an important factor in monitoring teams. Furthermore, the insight is useful in restricting the hypotheses space that must be maintained in reasoning about the activities of a team, as demonstrated by the computational savings we presented.

6. RELATED WORK

OVERSEER differs from most previous work on plan-recognition in being focused on monitoring multiple agents, not a single agent. A small number of previous studies have addressed this problem, and we discuss them below.

Like OVERSEER, previous work by Tambe [17] also focuses on explicitly using team intentions for inferring *team plans* from observations. However, OVERSEER uses a more advanced teamwork model (e.g., it can predict failure states and recovery actions), uses knowledge about procedures used by a team (i.e., communication decisions), and also explicitly reasons about uncertainty and time, allowing it to answer queries related to the likelihood of current and future team plans (issues not addressed in [17]).

Work such as [2, 8] focuses on explicitly addressing uncertainty in plan recognition in multi-agent contexts, but does not exploit explicit notions of teamwork. For instance, [2] uses pattern matching to recognize tactics in military operations. Similarly, [8] relies entirely on coordination constraints among agents to recognize team tactics. However, as is well known, teamwork is more than just simultaneous coordinated activity [4]. Thus, a purely coordination-based approach is likely to face difficulties in general (as acknowledged in [8]): If a team member were to suddenly fail, a pure coordination-based approach may fail to recognize the plan the team attempted to execute. In contrast, OVERSEER can predict role replacement and continue with its monitoring. An additional difference is that OVERSEER monitors a team using very limited observations—only of some agents, some of the time, while both [2] and [8] use observations of all agents, at all times.

Huber [7] demonstrates the use of probabilistic plan-recognition in coordinating with multiple agents, and its benefits in communication savings. Huber's work does not take into account any knowledge of relationship between agents, and does not focus on efficiency in representing multiple agents, in contrast to YOYO*.

A complementary line of work on TEAMCORE has demonstrated that plan-recognition-based monitoring can be used to dynamically adapt the communication patterns of agents, such that monitoring is made easier [20]. This work (i) reduced, but did not eliminate uncertainty, and (ii) did not present methods to address uncertainty, as we do here. However. it presents an interesting future direction for OVERSEER's development, where it takes active steps to improve its monitoring capabilities.

7. SUMMARY AND FUTURE WORK

We have presented OVERSEER, a system for monitoring previously deployed distributed teams. Monitoring of such teams is a difficult challenge, as one cannot rely purely on cooperative reports from the monitored agents: (i) A previously deployed system is difficult to modify such that agents communicate their state, especially when using off-the-shelf, proprietary, and legacy components; (ii) the monitored information may change based on the monitoring task, and it is unreasonable to expect developers to change the behavior of the agents for each such change; and (iii) as is well recognized in the literature, requirements for communication costs and reliability are prohibitive.

To address this challenge, OVERSEER, employs a novel technique which utilizes plan-recognition to infer agents' state from the observable *routine* communications. Such a technique is not intrusive, but suffers from large uncertainty, due to the limited number of observations available. To tackle this uncertainty, OVERSEER employs a number of novel techniques, which exploit knowledge of the relationships between the agents to alleviate uncertainty and increase efficiency of monitoring: (i) An efficient probabilistic algorithm for plan-recognition; (ii) YOYO*, an approach for efficient recognition of coherent hypotheses; and (iii) the use of team responses predictions to alleviate uncertainty. These techniques are general-purpose, and could be potentially applied to the growing number of distributed agent-based applications such as [1, 6, 16].

We provided an in-depth empirical evaluation of these techniques in one of the domains in which OVERSEER is applied. The evaluation carefully examines the contribution of each technique to the overall recognition success, and demonstrates that these technique work best together, as they complement relative weaknesses of each other. The paper also presented an evaluation of the expressivity–scalability trade-off in OVERSEER.

Acknowledgements

We thank Yves Lespérance and the anonymous reviewers for useful comments, and Manuela Veloso for her support of this project at CMU. As always, K. Ushi deserves many special thanks. This research was supported by DARPA awards F30602-98-2-0108, F30602-98-2-0135, and F30602-00-2-0549, and managed by the Air Force Research Labs/Rome site.

8. REFERENCES

[1] P. R. Cohen, M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow. Quickset: Multimodal interaction for distributed applications. In *Proceedings of the Fifth Annual International Multimodal Conference (Multimedia '97)*, pages 31–40, 1997.

[2] M. Devaney and A. Ram. Needles in a haystack: Plan recognition in large spatial domains involving multiple agents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 942–947, Madison, WI, 1998.

[3] T. Finin, Y. Labrou, and Mayfield. KQML as an agent communication language. In J. Bradshaw, editor, *Software Agents*. MIT Press, 1997.

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

[5] B. J. Grosz and S. Kraus. The evolution of sharedplans. In M. Wooldridge and A. Rao, editors, *Foundations and Theories of Rational Agency*, pages 227–262. 1999.

[6] B. Horling, V. R. Lesser, R. Vincent, A. Bazzan, and P. Xuan. Diagnosis as an integral part of multi-agent adaptability. Technical Report CMPSCI Technical Report 1999-03, University of Massachusetts/Amherst, January 1999.

[7] M. J. Huber and T. Hadley. Multiple roles, multiple teams, dynamic environment: Autonomous netrek agents. In W. L. Johnson, editor, *Proceedings of the International Conference on Autonomous Agents*, pages 332–339, Marina del Rey, CA, 1997. ACM Press.

[8] S. S. Intille and A. F. Bobick. A framework for recognizing multi-agent action from visual evidence. In *Proceedings of the National Conference on Artificial Intelligence*, pages 518–525. AAAI Press, July 1999.

[9] N. R. Jennings. Commitments and conventions: the foundations of coordination in multi-agent systems. *Knowledge Engineering Review*, 8(3):223–250, 1993.

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

[11] G. A. Kaminka and M. Tambe. Robust multi-agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research*, 12:105–147, 2000.

[12] C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, P. J. Modi, I. Muslea, A. G. Philpot, and S. Tejada. Modeling Web sources for information integration. In *Proceedings of the National Conference on Artificial Intelligence*, 1998.

[13] D. T. Ndumu, H. S. Nwana, L. C. Lee, and J. C. Collis. Visualizing and debugging distributed multi-agent systems. In *Proceedings of the International Conference on Autonomous Agents*. ACM Press, May 1999.

[14] NEON: New Era of Networks, Inc. Product: NEONet. http://www.neonsoft.com/.

[15] T. R. Payne, K. Sycara, M. Lewis, T. L. Lenox, and S. Hahn. Varying the user interaction within multi-agent systems. In *Proceedings of the International Conference on Autonomous Agents*, pages 412–418, 2000.

[16] M. Pechoucek, V. Marik, and O. Stepankova. Role of acquaintance models in an agent-based production planning system. In *Proceedings of the International Workshop on Cooperative Information*, 2000.

[17] M. Tambe. Tracking dynamic team activity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, August 1996.

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

[19] M. Tambe, W. L. Johnson, R. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), Spring 1995.

[20] M. Tambe, D. V. Pynadath, N. Chauvat, A. Das, and G. A. Kaminka. Adaptive agent integration architectures for heterogeneous team members. In *Proceedings of the International Conference on Multiagent Systems*, pages 301–308, Boston, MA, 2000.

[21] L. Vercouter, P. Beaune, and C. Sayettat. Towards open distributed information systems by the way of a multi-agent conception framework. In *Working Notes of the AAAI-2000 Workshop on Agent-Oriented Information Systems (AOIS-2000)*, pages 29–38, 2000.