A Fly on the Wall: Monitoring Agent Organizations by Eavesdropping

Gal A. Kaminka and **David V. Pynadath** and **Milind Tambe**
Information Sciences Institute
University of Southern California
{galk, pynadath, tambe}@isi.edu

Abstract

The increasing ubiquity of complex agent organizations has led to an increasing need for the on-line monitoring of such organizations. However, in many domains (e.g., those with information agents), we cannot observe the agents' actions. In addition, we rarely have the ability to change the agents themselves to force them to communicate their state to us. Fortunately, we can often eavesdrop on messages communicated by the agents as part of their natural coordination. This paper presents an approach for *on-line* monitoring of organizations using messages as observations. This approach includes the following key novel ideas: (i) a *linear time* probabilistic plan-recognition algorithm, particularly well-suited for processing communications in agent organizations; (ii) a technique for modeling the agent organization as a single coherent entity—trading expressivity for scalability; and (iii) an approach to exploiting general knowledge of teamwork to predict organizational responses during normal and failing execution, to reduce monitoring uncertainty. We present an empirical evaluation of these ideas in the context of monitoring a complex, multi-agent system.

1 Introduction

With the growth of applications involving multi-agent organizations, there is now an increasing interest in monitoring agent organizations by listening in on agent communications (Ndumu *et al.* 1999). Communication-based monitoring is important for several reasons. First, such monitoring is non-intrusive, not requiring agents to change to existing communications or other behaviors. Second, it is often difficult to directly observe a distributed set of heterogeneous agents (particularly software agents), so listening in on their communications is the main cost-effective monitoring technique available. Third, the growth of agent development and integration architectures (Ndumu *et al.* 1999), which standardize at least some aspects of agent communication, provide increasing opportunities for such monitoring. Such monitoring is useful in many multi-agent applications, e.g., (Horling *et al.* 1999).

Our work focuses on agent teams, where agents ask each other to jointly execute or terminate executing team plans. By listening in on such messages, the system must answer queries about the monitored organizations' current and future states. These queries may be about any unit of the organization, from the high-level team, through sub-teams, to the individuals. For instance, a query may check what plan(s) the high-level team is currently executing, to check if it is making adequate progress. Another query may check the future likelihood that a subteam will fail in fulfilling their role — to take remedial actions if this likelihood is high.

There are unfortunately several difficulties in accomplishing this goal. First, team members cannot in practice continuously communicate about all their on-going plans and actions (Grosz & Kraus 1996). Such communication results in significant uncertainty in inferring the teams' ongoing plans. Second, communications sometimes occur in small subteams, and yet the monitoring system must infer the state of the entire team and other subteams. Third, agents in teams may unexpectedly fail, which increases the uncertainty and requires that the system predict how a team responds to agent failure. An important final constraint is that the monitoring must occur on-line, i.e., the inference procedures must be efficient.

We have developed a system, called *Eavesdropper*, which addresses the above difficulties. Eavesdropper uses multi-agent plan recognition to monitor agent organizations based on their routine communications. While previous work in multi-agent plan-recognition has either focused on exploiting explicit teamwork reasoning, e.g., (Tambe 1996), or explicitly reasoning about uncertainty when recognizing multi-agent plans, e.g., (Intille & Bobick 1999), a key novelty in Eavesdropper is that it effectively blends these two threads together.

Eavesdropper combines the following novel techniques: First, it uses an efficient, *linear time* probabilistic plan-recognition algorithm, particularly well-suited for processing communications in agent organizations. Second, rather than modeling each individual agent separately, it models the agent organization as a single coherent entity — sacrificing some expressivity for efficiency and scalability. Finally, by exploiting general knowledge of teamwork, it can effectively predict (and hence effectively monitor) organizational responses during normal execution, as well as during a general class of failures. We have implemented this approach, and we present an evaluation of it in the context of monitoring a complex, multi-agent system.

2 Domains and Motivations

Concretely, Eavesdropper has been applied to teams of heterogeneous, distributed software agents, consisting of 10 to 20 team members. These teams are developed using the

Copyright © 2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

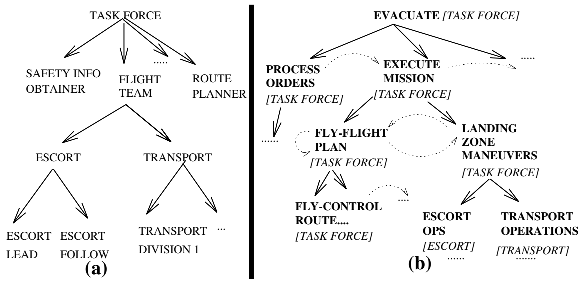

Figure 1: Portions of the team-hierarchy (a) and plan-hierarchy (b) used in our domain. Dotted line show temporal transitions.

Teamcore multi-agent integration architecture (Pynadath *et al.* 1999). As mentioned earlier, there is an increasing interest in such architectures to enable the growing numbers of heterogeneous software agents and smart hardware devices to work together to address large-scale problems (Huhns & Singh 1998). Teamcore accomplishes this integration via a distributed team of communicating proxies.

Each Teamcore proxy represents a single domain agent. The Teamcore proxies jointly execute 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 team-oriented 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. There are teams assigned to execute the plans, e.g., *Task Force* team jointly executes EVACUATE. To execute the team-oriented program, each proxy works with an in-built domain-independent teamwork model, called Steam (Tambe 1997). Steam allows the agents to automatically coordinate using selective communications.

Humans and agents must monitor the resulting team, querying about the present and future likely states of the entire team, its subteams and individuals—to monitor progress, compute likelihoods of failure, etc. Eavesdropper non-intrusively performs such monitoring, listening in on agent communications remotely (over the Internet) and inferring the state of the team. This is a challenging task because: (i) only some agents communicate, and only about some plans; and (ii) agents may occasionally fail, but Eavesdropper cannot observe these failures. Thus Eavesdropper faces significant uncertainty in its monitoring.

We have applied the Teamcore framework to the problem of rehearsing the evacuation of civilians from a threatened location. An integrated system must enable a human commander to interactively provide locations of the stranded civilians, safe areas for evacuation and other key points. Simulated helicopters fly a coordinated mission to evacuate the civilians. The integrated system must itself plan routes to avoid known obstacles, dynamically obtain information about enemy threats, and change routes when needed.

The application integrates a set of diverse agents: Quickset (Multi-modal command input agents, OGI), Route planner (Path planner for aircraft, CMU), Ariadne (Database engine for dynamic threats, USC/ISI), and 8 Helicopter pilots (Pilot agents for simulated helicopters , USC/ISI). The system has 11 corresponding Teamcore proxies, which execute a team-oriented program consisting of 40 team plans. The teamcores exchange about 100 messages, while also communicating with the domain agents.

3 Plan Recognition of Individual Agents

In domains such as those mentioned previously, we can perform some prediction of runtime execution by reasoning individually about each of the team members using a plan hierarchy. We also have access to certain coordination messages sent by each agent indicating certain mileposts in plan execution (usually initiation or termination). This section presents a mechanism that uses only the messages sent by a single team member for recognizing its plans.

For instance, if we observe a message about the initiation of FLY-FLIGHT-PLAN, then we know from Figure 1 that PROCESS-ORDERS cannot be a possible future state of the agent. Both FLY-FLIGHT-PLAN and LANDING-ZONE-MANEUVERS *are* possible future states, but the recognition system has no basis for differentiating between the two. We address this ambiguity through a probabilistic model that supports quantitative evaluation of the hypotheses. We use a time series of state variables, where, at each point of time, the agent's state is the set of plans it is currently executing. We represent these plans by a set of boolean random variables, $\{X_t\}$, where each variable X_t is true if and only if plan X is active at time t.

We can represent our beliefs about the agent's actual state at time t as a probability distribution over all variables $\{X_t\}$. We begin with a certain belief that the agent is executing its top-level plan at time 0. We can propagate this belief throughout the hierarchy using the method described in Section 3.1 to simulate plan execution with the passage of time. When we observe messages, we can incorporate the evidence into our beliefs according to the method described in Section 3.2.

3.1 Belief Update When No Message Is Observed

If we do not observe communication, then we roll the model forward to the next time slice. For each plan that the agent could be executing, we must compute 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 $e^{-\lambda_X \tau}$. The parameter λ_X then corresponds to 1/(average duration of X). Given this model of plan duration, the probability of the plan's completion between times t and $t+1$ is simply $\Pr(\mathit{done}(X,t)|X_t) = 1 - e^{-\lambda_X}$.

Once we use the exponential model to compute the probability of plan termination, we then need to determine which plan the agent will execute next. We examine all of the possible successors and compute the probability of taking the corresponding transition, conditioned on the fact that no message was sent. For each plan, X, we record the probability of entering each successor, Y, given that X has just completed: $\pi_{xy} = \Pr(Y_{t+1}|X_t, done(X,t))$. We also record the probability of seeing a message given the transition, $\mu_{xy} = \Pr(msg_t|X_t, Y_{t+1})$. In both cases, we make a Markovian assumption that the plan history before time t does not affect the probabilities. We can potentially obtain the three sets of parameters, λ, μ, and π, from domain experts or from frequency counts over previous executions. We can now combine these values to get the desired conditional probability:

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

The normalizing denominator, η_X, is simply the sum of the numerator over all possible successors, Y. We can precompute these sums offline. 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.

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. We compute the probability of transitions out of the parent plan as of the child, except with this new completion probability replacing the exponential distribution.

If the plan has children, then we must also distribute the incoming probability among them. Since we assume that all plans take at least a single time step to complete, we consider only the first children. In Figure 1, upon first entering the top-level plan EVACUATE, the only possible child plan that can be active at time 0 is PROCESS-ORDERS. We compute the probability over multiple first children by dividing the probability incoming to the parent among them. If any of these children have child plans of their own, this new incoming probability is distributed in turn, using the same method. 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 ∈ M do
  b_{t+1}(X,¬block) += b_t(X,¬block) e^{−λ_X}
  if η_X = 0 then {Message required}
    b_{t+1}(X,block) ← b_t(X,¬block)(1 − e^{−λ_X}) + b_t(X,block)
  else {Message not required}
    for all plans Y that succeed X do
      ρ ← b_t(X,¬block)(1 − e^{−λ_X})(1 − μ_xy) π_xy / η_X
      if Y = done then
        b_{t+1}(parent(X), block) += ρ
      else {Y is a sibling plan}
        b_{t+1}(Y,¬block) += ρ
        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 ∈ M consistent with m do
  if m is an initiation message then
    b'(X,¬block) ← b_t(X,¬block)
  else {m is a termination message}
    for all plans Y ∈ M that succeed X do
      b'(Y,¬block) ← b_t(X,block) μ_xy π_xy / (1 − η_X)
normalize distribution b'
for all plans X ∈ M with b' > 0 do
  b_{t+1}(X,¬block) ← b'(X,¬block)
  b_{t+1}(parent(X),¬block) += b'(X,¬block)
  PROPAGATE-DOWN(X, b'(X,¬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 and space complexity linear in the number of plans and transitions in M. We gain this efficiency from two sources. First, the assumption of a memoryless exponential distribution over plan duration 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 Knowledge

The previous section has presented an efficient method for probabilistic recognition of an individual agent's state. A naive extension of this approach to multi-agent plan-recognition is to use an array of such individual models, where each one of them gets updated as observations come in. Queries about the state of a team are then answered by looking at the state of the members which make up the team.

This approach proves insufficient when monitoring a team via its communications. A key difficulty lies in the scarcity of observations: Realistically, agents communicate only in particular circumstances. Furthermore, often not all the agents communicate—sometimes a single message from one agent is intended to change the state of all its listeners (for instance, a plan termination message). This results in uncertainty about the state of agents and subteams which do not communicate, or communicate rarely, since no observations are available about them.

This section reports on several novel techniques intended to improve recognition accuracy in limited-observations settings. The techniques builds on using team and teamwork knowledge as the basis for disambiguating the recognition hypotheses.

4.1 YOYO*: Efficient 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 phenomena 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, etc. It is called *Team-coherence* (Kaminka & Tambe 1999).

As coherence is important in teams, we can use it as a heuristic, preferring recognition hypotheses in which team-members are in the same state over hypotheses in which they are in different states. This 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. For example, when a TERMINATE-PLAN message is overheard, the recognition system could assume that the team-members have received the message, and terminated the plan jointly with the sender. Here, the system is preferring a hypothesis in which the team remains coherent (synchronized) over hypotheses in which team is incoherent.

We present here the YOYO* algorithm (Algorithm 3), an efficient technique for reasoning about coherent hypotheses. YOYO* relies on a single plan-hierarchy that is fully expanded for all subteams, and an algorithm which climbs up the team- and plan-hierarchies to then go down subtrees and re-align prior knowledge such that it is consistent with the new observation. This technique trades expressivity for efficiency (see Section 5).

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

Loop forever:
if messages rec'd–new plan S team T, **then**
  Incorporate-Evidence(T, S)
  $tmp \leftarrow T$
  while tmp is not the root team in H **do**
    find in M lowest common ancestor A of S joint to tmp and its sibling teams
    for each child transition of A whose team≠T **do**
      SCALE(the subtree roots at the child), so its state probabilities sum up to the new probability of A
    $tmp \leftarrow parentof(tmp)$
else
  PROPAGATE-FORWARD in M

The key idea in YOYO* is that once a step up is taken in the team- and plan-hierarchies, 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 must now take a team T into account when incorporating evidence: Only transitions that T is allowed to take may be followed. Propagate 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.

For example, suppose that the entire team is known to be executing FLY-FLIGHT-PLAN. Now, a message exchange is observed among the members of the TRANSPORT team, indicating that it has begun execution of TRANSPORT-OPS. First, the new evidence is incorporated for the transport team. Among other changes, the probability of the plan LANDING-ZONE-MANEUVERS (LZM) 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 LZM 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. Section 5 provides an in-depth evaluation.

4.2 Predicting Organization Responses

We explore here an independent technique for improving recognition success. The technique predicts when communications will take place during task execution. In addition, it predicts the likelihood 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 decisions 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 hypotheses. For instance, if an establishment message is expected as agents take the transition from FLY-FLIGHT-PLAN to LZM (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 the latter.

The knowledge required for such organization responses predictions can be acquired by learning, and/or by projecting the state of the team into the future, using a teamwork model to determine how the team will respond given its (future) state. We have discovered that simple rote-learning was sufficient in many cases to build a very effective model of the communication policy of an organization. However, such an approach is limited when it learning failure probabilities, and failures happen infrequently.

A different approach altogether is to dynamically generate predictions of the responses based on the known state of the team, and a teamwork model which is used to simulate the responses of the team given likely future states. A domain-dependent teamwork model, such as STEAM, can be re-used here for monitoring purposes. Such a model is usually used to generate decisions on when and how to communicate, or how the coordination relationships required by the designer are maintained. However, it can also be used in simulation: Future states and transitions are fed in as if they are the current state, and the teamwork model's decisions are used to predict the responses taken by the monitored team. This approach is dynamic and allows us to incorporate the state of team-members at run time, however it provides approximate results, due to the uncertainty in the simulation.

Dealing with failures: Eavesdropper uses the above mechanism to utilize known coordination relationships among the agent and the agent's known state to predict the failure probabilities of plans. For instance, if a plan requires that at least one agents in a subteam participate, Eavesdropper is able to dynamically compute the probability that the plan will fail based on the individual failure probabilities. Eavesdropper can also simulate the passage of time in the system, to detect delayed, failed, or out-of-order messages.

5 Results and Evaluation

Eavesdropper is a fully implemented system, which we have been using in actual runs of the system over the internet. The first part of our evaluation tests the contribution of the different techniques in Eavesdropper to recognizing the correct state of the agents and teams. Figure 2 compares the average accuracy for a sample of our actual runs, where accuracy is measured as picking the correct hypothesis as the most likely hypothesis. The runs are marked 'A', 'B', 'C' and 'D' (X-axis). The average accuracy is given in the 0-1 range.

The average accuracy when using the individual models with no coherence (Section 3) is presented in the leftmost bar in each group (Figure 2), and is clearly very low. The next bar presents the expected average recognition accuracy if only coherence is used to rule out hypotheses (Section 4.1), and then an arbitrary selection is made among the remaining hypotheses. Coherence is clearly an effective constraint, as it brings the accuracy up by almost 15% without using any probabilistic reasoning. The next bar to the right (Coherent, Temporal) presents the results of combining both coherence and the probabilistic reasoning capabilities, including the model of plan durations and the known failure probabilities (Sections 3 and 4.1). This approach offers a significant boost in accuracy, when compared to the approach relying on either information alone. The remaining bar presents the average accuracy in each run using coherence, the duration and failure probabilities, as well as the

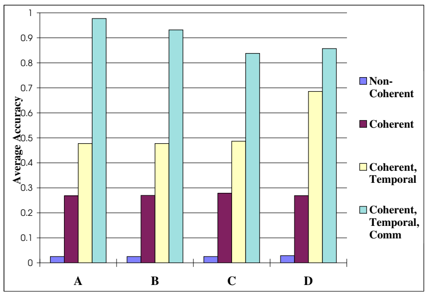

Figure 2: Average accuracy in sample runs.

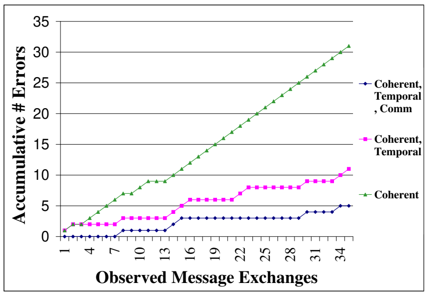

Figure 3: Accumulative number of errors in run 'D'.

team communication model, which predicts when communications will occur (Sections 3–4). The results demonstrate that while a model of plan duration, and plan-failures, can certainly improve the accuracy of the recognition, a significant boost in accuracy is achieved through the ability to predict at what point during task execution communication will occur. In many cases, it was able to rule out all hypotheses but one or two. The accuracy results within each run were analyzed using ANOVA and found to be statistically significant with confidence greater than 99.999999%.

Figure 3 shows the system in action in a particular run (D above). The figure presents the accumulative number of errors as time goes by during the run, where an error is defined as a failure to choose the correct hypothesis is the most likely one (i.e., 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, establishing or terminating a plan. The line marked *Coherent* shows the accumulative 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 run using both coherence and the plan-durations model to choose the most likely hypothesis. This approach is much more successful, as evident by the slower rise in the number of accumulative errors. Finally, the remaining line displays the results of using coherence, the plan-duration model, and the model predicting communications. This results in slightly less than half the accumulative errors of the coherent-temporal approach.

The second part of evaluating Eavesdropper examines a key trade-off 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 to using it, strictly ruling out reasoning about incoherences. It is impossible for YOYO*, for instance, to represent an incoherence in which two subteams, in service of a common team, are in disagreement about what the plan is executed by the common team. It is also difficult to explicitly represent hypotheses associated with communication losses and delays, which also cause incoherence. YOYO* is still able to detect many incoherences–it would fail to find the states referred to in the messages and announce failure–but is not be able to represent hypotheses explaining the incoherences, and it is not guaranteed to detect all potential failures. An approach in which each individual is represented separately allows for such representation, and in this respect is more expressive.

On the other hand, YOYO* offers great computational advantages when compared to the individual representation approach. YOYO* requires a single, fully-expanded, task-model to represent the entire team. This model is a union of all the individual agent task-models. In the worst case, every agent carries out a completely different plan. In this case, YOYO* will be as inefficient as the individual recognition approach, requiring as much space. However, a more typical case is where agents to share many plans and transitions, because they carry them out jointly. In this case, YOYO* offers significant savings. The best case is when a homogeneous team is jointly executing all tasks, with no subteams or individual roles. Then the reduction in space complexity is from N models to 1. *In our application domains the space savings are about 86%*: 66 plan-nodes were used in YOYO*, 496 nodes were used in individual models, with proportional time savings.

The reduction in the space requirements leads to appropriate reduction in computation time as well. Probabilities are propagated once for the entire team, and do not need to be propagated repeatedly for each individual. Furthermore, YOYO* represents only a linear number of hypotheses (in the size of the plan-hierarchy), because it rules out most incoherent hypotheses. The individual-models approach, however, represents (implicitly) an exponential number of hypotheses. Therefore, tasks that require enumeration of hypotheses will have exponential running times when using individual models.

6 Related Work

Like Eavesdropper, previous work by Tambe (Tambe 1996) also focuses on explicitly using team intentions for inferring *team plans* from observations. However, Eavesdropper uses a more advanced teamwork model (e.g., it can predict failures based on coordination constraints), and also explicitly reasons about uncertainty, allowing it to answer queries related to the likelihood of team plans.

The general probabilistic plan-recognition approach of (Charniak & Goldman 1993) could explicitly reason about uncertainty in multi-agent systems. However, the potentially unbounded number of observed communication in the target domains would render the approach impractical for online recognition. Work such as (Devaney & Ram 1998; Intille & Bobick 1999) focuses on explicitly addressing uncertainty in plan recognition in multi-agent contexts, but does not exploit explicit notions of teamwork. For instance, (Devaney & Ram 1998) uses pattern matching to recognize tactics in military operations. Similarly, (Intille & Bobick 1999) relies entirely on coordination constraints among agents to recognize team tactics. However, teamwork is more than just simultaneous coordinated activity. Thus, a purely coordination-based approach is likely to face difficulties in general, as acknowledged in (Intille & Bobick 1999): For instance, if a team member were to suddenly fail. In contrast, Eavesdropper can predict role replacement and continue with its monitoring. Eavesdropper can also monitor a team using limited observations.

7 Summary and Future Work

We have presented a system for monitoring organizations via their communications. Key novelties include: An efficient probabilistic algorithm for plan-recognition; YOYO*, an approach for efficient recognition of coherent hypotheses; and the use of organization responses predictions to alleviate uncertainty. We provided an in-depth empirical evaluation of these techniques, carefully showing the contribution of each to the overall recognition success. Eavesdropper gains its efficiency at the cost of several assumptions about the nature of the domain. We are currently exploring ways to relax these assumptions without incurring significant computational costs.

References

Charniak, E., and Goldman, R. P. 1993. A Bayesian model of plan recognition. *Artificial Intelligence* 64(1):53–79.

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

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

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

Huhns, M. N., and Singh, M. P. 1998. All agents are not created equal. *IEEE Internet Computing* 2:94–96.

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

Kaminka, G. A., and Tambe, M. 1999. I'm OK, You're OK, We're OK: Experiments in distributed and centralized social monitoring and diagnosis. In *Proceedings of the International Conference on Autonomous Agents*. Seattle, WA: ACM Press.

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

Pynadath, D. V.; Tambe, M.; Chauvat, N.; and Cavedon, L. 1999. Toward team-oriented programming. In *Proceedings of the*

Agents, Theories, Architectures and Languages (ATAL'99) Workshop (to be published in Springer Verlag "Intelligent Agents V"), 77–91.

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

Tambe, M. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR)* 7:83–124.