EXECUTION MONITORING IN MULTI-AGENT ENVIRONMENTS

by

Gal A. Kaminka

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

(Computer Science)

May 2000

Copyright 2007 Gal A. Kaminka

EXECUTION MONITORING IN MULTI-AGENT ENVIRONMENTS

by

Gal A. Kaminka

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Science)

May 2000

Copyright 2007 Gal A. Kaminka

Dedication

To my grandma Lilly, who taught me that science is a question.
To my parents, Shlomit and Nachum, who taught me that time is precious.
To my wife Oshra, who taught me that faith is a necessity.
To my children, who taught me that miracles are common.

Acknowledgments

When I first came aboard to the USC Computer Science Ph.D. program, little did I know what I was in for. My idea of the Ph.D. program was largely based on my undergraduate degree experience: I was more or less thinking that getting the Ph.D. would involve taking a few tough classes, and writing a really long paper... Boy, was I wrong!

In the statement of purpose which I had written as part of the application process, I had written that I need the room to spread my wings, so I would learn to fly. I now know that learning to fly is not so much as issue of room. It's much more an issue of finding teachers who will show you how to spread your wings. I am very fortunate to have been offered the opportunity to work with good teachers. My only part in all of this was to listen to what they said.

First and foremost among these teachers is my advisor, Milind Tambe. His devotion to his students, his care in teaching them science rather than merely getting them through the program, his insistence on pushing their abilities to the limit, his taking great care in providing students with what they need to do their research, his ethical standards, and his accessibility combine to provide the best advisor I could have asked for. Never did Milind show anything but careful consideration of any ideas I came up, though God (and Milind, and me) knows how silly some of them have been. He was patient and took it as given that it was part of his job to listen to my occasional never-ending ramblings, and convince me, rather than tell me, of anything that needed more thought.

My committee members deserve many thanks for their advice and friendliness. I had the good fortune of having some of the best people in my field on my committee, and this dissertation is much better because of them. George Bekey, and Dan O'Leary made themselves available to provide help and advice, and made sure that I saw the big picture when it came to the research. They carefully considered

ideas I presented, and were quick to help me generalize and extend them. Victor Lesser has encouraged me and supplied me with tons of useful pointers and comments. He helped me clarify the assumptions underlying the work, and spent much time making sure I understand what it is that I'm doing. Jeff Rickel made sure I say what I want to say clearly, and that I say it in English. But really, he has been extremely instrumental in my thinking about the research. He was helpful in pointing out the limits of the techniques I presented, and deserves much credit for making me think about the research from completely fresh perspectives. I owe him a great many thanks.

Yolanda Gil, Daniel Marcu, Stacy C. Marsella, Wei-Min Shen, Onn Shehory, and Jihie Kim all provided useful advice and the benefit of their wisdom and experience in carrying out a research program, making it in time through the Ph.D. process, tackling research problems, writing papers, and making career decisions. David V. Pynadath served as a role-model for the job-search process, as well as for finishing up the program under pressure. His advice and willingness to listen proved invaluable. Ion and Maria Muslea, Irene Langkilde, Hyuckchul Jung, Paul Scerri, and Yaser Al-Onaizan are all good friends who provided important moral support at difficult times, and shared the ups and downs of the PhD program. My never ending discussions with Jose-Luis on everything from the semantics of multi-valued logic through socio-economic problems to the merits of different government systems have cost us both much research time (to both our wives' occasional dismay), which I thoroughly enjoyed.

Right when I started at USC, I was joined and supported by the two other Musketeers, Sheila Tejada and Jafar Adibi. Without Sheila's invitation to join them for a study group in CSCI-598, I would not have made it through my first semester. Without their continued friendship and support, I would not have made it through the program with my sanity. It is really as simple as that. They both have taught me so much. I've learned from Sheila about creating the atmosphere for teams that can do serious work in a fun way. From Jafar I have learned that friendship can override artificial political borders; People are people.

I used to read acknowledgment sections of dissertations I came across, always thinking what I would be writing when it came to my own section. To my great surprise, I find myself writing pretty much what others have written before me, no

doubt because ultimately, while the experience is different for each one of us going through a Ph.D. program, the ingredients are the same. Indeed, one final ingredient is often mentioned by others in their dissertations, and I have yet to mention it in mine.

My family has been a source of strength and wisdom during the Ph.D. years. My parents and in-laws, my real and in-law siblings, our adopted parents in Los Angeles, have all chipped in, occasionally coming from overseas to lend a hand and offer a shoulder. My grandmother, who has passed away during the time I spent at USC, deserves special recognition. From the time I was a toddler, and continuing until the time of her death, she insisted on the need for constant questioning, thinking, answering. She was the first to make me think about thinking, which ultimately resulted in my pursuing a degree in Artificial Intelligence.

To say that my wife Oshra has helped me get through would be a terrible understatement. She has carried me through rough times, made sure I see things in perspective, and gave of herself to make sure this dissertation actually gets written, all while working and finishing up her own graduate degree. I would be lost without her, and my greatest motivation has always been to be worthy of her love. Together, we had two boys, Dor and Tom, while our graduate studies progressed. I am often asked whether they did not influence our abilities to do well at school. I always answer in a resounding yes—they definitely helped.

Contents

Dedication iii

Acknowledgments iv

List Of Figures x

List Of Tables xi

List Of Algorithms xii

Abstract xiii

1 Introduction 1
- 1.1 Monitoring Selectivity in Relationship Failure Detection 5
- 1.2 Monitoring Selectivity in Monitoring Distributed Teams 7
- 1.3 Organization of This Thesis 8

I Monitoring for Relationship Failure Detection 10

2 Motivation and Background: Failure Detection 11

3 Socially-Attentive Monitoring for Failure Detection 14
- 3.1 A Knowledge-Base of Relationship Models 14
- 3.2 Knowledge of Monitored Agents and Team 16
  - 3.2.1 Representation 16
  - 3.2.2 Acquisition 17
- 3.3 Relationship Violation Detection 20
- 3.4 Relationship Diagnosis 24

4 Monitoring Selectivity in Centralized Teamwork Monitoring 27

5 Monitoring Selectivity in Distributed Teamwork Monitoring 39

6 Using Socially-Attentive Monitoring in an Off-Line Configuration 45

7 Beyond Teamwork Failures 51
7.1 A Richer Agreement Model: Agreeing to Disagree 51
7.2 Monitoring Using Role-Similarity Relationships 53

II Monitoring Distributed Teams 55

8 Motivation and Background: Monitoring Distributed Teams 56

9 Exploiting Teamwork Knowledge for Accurate Monitoring 62
9.1 Three areas of uncertainty in monitoring teams 63
9.2 Using knowledge of relationships 65
9.2.1 Using Teamwork: Revisiting Coherence 66
9.2.2 Using the team-hierarchy and knowledge of roles 68
9.3 Predicting Team Responses 68
9.3.1 Using predicted communications 69
9.3.2 Predicting team-responses to failure 72
9.4 Accuracy Evaluation 73

10 Scaling-Up Monitoring Efficiency 79
10.1 Optimizing using the team-hierarchy 79
10.2 YOYO*: Efficiently reasoning about coherent hypotheses 82
10.3 Efficiency Evaluation 85

11 Related Work 88
11.1 Teamwork 88
11.2 Multi-Agent plan recognition 91
11.3 Multi-Agent Monitoring Selectivity 94
11.4 Monitoring in Single-Agent Settings 96
11.5 Diagnosis 98
11.6 Other Related Work 100
11.7 Summary 103

12 Conclusions and Future Work 104
12.1 Contributions 105
12.2 Future Work 106
12.3 Acknowledgment 108

Bibliography 109

Appendix A
List of Selected Publications 118

Appendix B
Proofs 120

Appendix C
Efficient Single-Agent Probabilistic plan recognition 123
C.1 Belief Update When No Message Is Observed 123
C.2 Belief Update with Observed Message 124
C.3 Individual Agent Recognition Complexity 126

Appendix D
Socially-Attentive Monitoring Algorithms 127
D.1 RESL 127
D.2 Detection of Failure, Centralized and Distributed Teamwork Monitoring 127
D.3 Probabilistic version of YOYO* 129

List Of Figures

1.1 Organization of this Dissertation. 9

2.1 A plan-view display (the ModSAF domain) illustrating the failure in Example1. The thick wavy lines are contour lines. 12

3.1 The general structure of a socially-attentive monitoring system. . . . 15
3.2 Portion of Hierarchical Reactive Plan Library for ModSAF Domain (Team plans are boxed. These are explained in Section 3.3). 18
3.3 Scout (a) and Attackers' (b, c) actual and recognized *abbreviated* reactive plan hierarchies. 19
3.4 Comparing two hierarchical plans. The top-most difference is at level 2. 21

6.1 An illustration of a switch. The three agents switch from plan 1 to plan 2. 46
6.2 ATA values for the Goalies sub-teams in games against Andhill'97. . . 49

7.1 A Portion of the plan hierarchy used by ISIS RoboCup agents. 52

8.1 Portions of the team-hierarchy (a) and plan hierarchy (b) used in our domain. Dotted line show temporal transitions. 58
8.2 Array of single-agent recognizers—one for each agent. 59

9.1 Learning of Communication Decisions 72
9.2 Average accuracy in sample runs. 75
9.3 Accumulative number of errors in runs 'A' and 'D'. 77

10.1 Array of single-agent recognizers, optimized (the plan-library changes based on agent's role). 81

List Of Tables

4.1 All possible failure permutations of the broken radio-link scenario (Example 2). 28
4.2 Scout's (A3) monitoring results in all permutations of Example 2. . . 29
4.3 All failure permutations of the undetected way-point scenario (Example 1). 31
4.4 Attacker's (A1) monitoring results in all permutations of Example 1. 32
4.5 Attacker's (A1) monitoring results in all permutations of Example 1, using team-*incoherence*. 34
4.6 A portion of the attacker's (A1) monitoring hypotheses and implied results when no ranking is used to select a single hypothesis for each case. 37

5.1 Scout's (A3) monitoring results in all permutations of Example 1, using team-coherence. 40
5.2 Observable partitioning of the helicopter pilot team in ModSAF. . . . 42

6.1 Average-Time-to-Agreement (ATA) for games against Andhill'97. . . 47
6.2 ISIS'97 and ISIS'98 mean score difference against Andhill'97, with changing communications settings 48

10.1 Trade-offs in monitoring efficiency versus expressivity 87

List of Algorithms

10.1 YOYO*(**plan hierarchy** M, **team-hierarchy** H) 84
C.1 PROPAGATE-FORWARD(**beliefs** b, **plans** M) 125
C.2 INCORPORATE-EVIDENCE(**msg** m, **beliefs** b, **plans** M) 125
D.1 RECOGNIZE(**agent** A) 128
D.2 DETECT($hierarchies_1, hierarchies_2, policy$) 128
D.3 CENTRALIZED-MONITORING 129
D.4 YOYO*(**plan hierarchy** M, **team-hierarchy** H) 130

Abstract

Agents in dynamic multi-agent environments must monitor their peers and the environment to execute individual and group plans, to ascertain their progress, and to detect failures. In practice, however, agents cannot continuously monitor all surroundings and their peers. This leads to uncertainty about monitored agents' states, and aggravates computational requirements. A key open question is thus how to limit monitoring activities while providing effective monitoring: *The Monitoring Selectivity Problem.* We investigate this question in the context of monitoring in teams of cooperating agents, in three complex, dynamic multi-agent domains, and in service of different monitoring tasks: Monitoring for coordination and teamwork failures, and monitoring distributed teams via their communications.

We provide empirical and analytical answers to the monitoring selectivity problem, via *Socially-Attentive Monitoring,* which focuses on using knowledge about the relationships between monitored agents and the procedures used to maintain these relationships. We explore a family of socially-attentive teamwork failure-detection algorithms under varying conditions of task distribution and uncertainty. We show that a centralized scheme using a complex algorithm trades correctness for completeness and requires monitoring all teammates. In contrast, a simple distributed teamwork monitoring algorithm exploits agents' local state and results in correct and complete detection of teamwork failures, despite relying on limited, uncertain knowledge, and monitoring only key agents in a team. In monitoring a distributed team, we present heuristics for tackling monitoring uncertainty (which results from the limited overheard communications), and provide empirical results demonstrating that socially-attentive techniques can significantly reduce the uncertainty in such monitoring. Furthermore, we explore monitoring algorithms which trade-off efficiency for expressivity, resulting in a limited-expressivity algorithm that can detect failures and provide high-accuracy monitoring of a team and its members, using a single, constant-space, structure. In addition, we report on the design of a socially-attentive monitoring system and demonstrate its generality in monitoring several coordination relationships, in providing quantitative teamwork evaluation, and in diagnosing detected failures.

Chapter 1

Introduction

Agents in complex, dynamic, environments face uncertainty in the execution of their tasks. Their sensors and actuators may fail unexpectedly, or their plans may fail due to the dynamic nature of the environment. For instance, a fog may hide normally-visible objects, mechanical failures may render a robot's cameras useless, a grip may be too slippery, etc. Interactions between agents in multi-agent environments add to the difficulty of execution, e.g., communication messages may get lost or garbled, and adversaries may intentionally interfere with plans. It is often expected that the agent's designer (or an automated planner) would provide the agent with sufficient knowledge on how to deal with this uncertainty in a preventative manner (for instance, reduce speed while driving in fog, use error correcting, secure communication protocols, etc.). This design methodology focuses on *pre-failure robustness* (Toyama & Hager, 1997): The agent's *plans* are made robust in face of uncertainty in the environment.

Pre-failure robustness, however, is inadequate by itself in many real-world environments. As application environments increase in complexity and unpredictability, the combinatorially explosive number of possible states makes it practically impossible for the agents' designer to predict at design-time all possible states the agents may find themselves in. Examples of such environments include (but are not limited to):

- Virtual environments for training and research (Rickel & Johnson, 1999a, 1999b; Kitano, Tambe, Stone, Veloso, Coradeschi, Osawa, Matsubara, Noda, & Asada, 1997).

- High-fidelity distributed simulations (Calder, Smith, Courtemanche, Mar, & Ceranowicz, 1993; Tambe, Johnson, Jones, Koss, Laird, Rosenbloom, & Schwamb, 1995).
- Intelligent homes (Horling, Lesser, Vincent, Bazzan, & Xuan, 1999).
- Multi-agent robotics (Parker, 1993; Balch, 1998).
- Application integration architectures and distributed applications (Pynadath, Tambe, Chauvat, & Cavedon, 1999; Pechoucek, Marik, & Stepankova, 2000).

In such environments the designer will therefore be unable to supply the agents with complete a-priori knowledge about the correct response in all states (Ambros-Ingerson & Steel, 1988). This difficulty is further exacerbated in environments in which multiple agents interact, since the designer is required to also predict all possible interactions and failures of interactions between the agents. For instance, a designer would need to predict when communications will fail between two cooperating agents, what an opponent's behavior will be, when sensors will fail, etc.

Agents in complex, dynamic, multi-agent environments must therefore be able to monitor their environment at execution-time, to detect any failures (leading to diagnosis and recovery (Toyama & Hager, 1997; Ambros-Ingerson & Steel, 1988; Wilkins, 1988)), to ascertain progress, and to track their own (and others') state (e.g., for visualization and debugging (Ndumu, Nwana, Lee, & Collis, 1999)). This process is called execution monitoring (Doyle, Atkinson, & Doshi, 1986; Ambros-Ingerson & Steel, 1988; Cohen, Amant, & Hart, 1992; Reece & Tate, 1994; Atkins, Durfee, & Shin, 1997; Veloso, Pollack, & Cox, 1998).

Execution monitoring in multi-agent settings requires an agent to monitor its peers, since its own correct execution depends also on the state of its peers (Cohen & Levesque, 1991; Jennings, 1993; Parker, 1993; Jennings, 1995; Durfee, 1995; Grosz & Kraus, 1996; Tambe, 1997; Pechoucek et al., 2000). Monitoring peers is of particular importance in teams, since team-members rely on each other and work closely together on related tasks:

- Monitoring allows team-members to coordinate their actions and plans with teammates, to help teammates and cooperate without interference. For example, drivers of cars in a convoy cannot drive without monitoring other cars in

the convoy, so as to not disband the convoy, and to help other drivers if cars break down.

- Monitoring allows team-members to use peers as dynamic information sources, for learning new information. For instance, if a driver in a convoy sees that the other cars in front of it suddenly turn to the left, she can infer the existence of an obstacle or milestone despite not directly seeing it herself.

Previous work in multi-agent monitoring has investigated different ways of monitoring in the context of teams of cooperating agents. For example, theoretical work on SharedPlans (Grosz & Kraus, 1999) has distinguished between *passive monitoring*, in which an agent is notified when a proposition changes (e.g., via communications), and *active monitoring*, in which an agent actively seeks to find out when a proposition changes (e.g., via observations and inference of unobservable attributes). Practical implementations have investigated the use of passive monitoring via communications (e.g., (Jennings, 1995), (Tambe, 1997)), active monitoring via plan recognition (e.g., (Huber & Durfee, 1995), (Intille & Bobick, 1999)), active implicit monitoring via the environment (Fenster, Kraus, & Rosenschein, 1995), and different combinations of these methods (Parker, 1993; Jennings, 1993; Tambe, 1997; Lesh, Rich, & Sidner, 1999). No approach has been shown to be clearly superior to another.

Regardless of the monitoring method, a key problem emerges when monitoring multiple agents: A monitoring agent must be selective in its monitoring activities (both raw observations and processing), since bandwidth and computational limitations prohibit the agent from monitoring all other agents to full extent, all the time (Jennings, 1995; Durfee, 1995; Grosz & Kraus, 1996). However, selectivity in monitoring activities leads to uncertainty about monitored agents' states, which can lead to degraded monitoring performance. We call this challenging problem the *Monitoring Selectivity Problem*: Monitoring multiple agents involves overhead that hurts performance; but at the same time, minimization of the monitoring overhead can lead to monitoring uncertainty that also hurts performance.

Although the monitoring selectivity problem has been raised in the past (see Chapter 11 for detailed treatment of related work), only solution frameworks and minimal constraints were provided (Jennings, 1993; Durfee, 1995; Grosz & Kraus, 1996). Key questions remains open:

- What are the bounds of selectivity that still facilitate effective monitoring?
- How can monitoring *accuracy* be maintained in face of limited knowledge of other agents' states?
- How can monitoring be carried out *efficiently* for on-line deployment?

For instance, the theory of SharedPlans requires agents to verify that their intentions do not conflict with those of teammates in order to guarantee the quality of teamwork (Grosz & Kraus, 1996). However, the practical methods by which such verification can take place are left for further investigation (Grosz & Kraus, 1996, p. 308). The designer is left with no concrete answer as to how to satisfy this constraint given the monitoring selectivity problem.

This dissertation begins to address the monitoring selectivity problem in teams by investigating requirements for effective monitoring in two monitoring tasks: Detecting failures in maintaining relationships, and determining the state of a distributed team (for both failure detection and visualization). Our focus in these explorations is on practical algorithms that have requirement and performance guarantees in real-world applications. The algorithms we present are *active* (per the definition above), in that they seek to identify the state of monitored agents via unintrusive key-hole plan recognition (i.e., without the active participation of the monitored agent in the monitoring process), as an alternative to passive monitoring (which may often be unreliable, as it intrusively requires the monitored agents to actively cooperate with the monitoring agent). However, we do not rule out the use of communications—we simply seek to provide techniques that can work even when communications-based monitoring fail.

We show how the monitoring selectivity problem emerges in the above two monitoring tasks, and we explore in-depth answers to it. We show that using knowledge about the ideal relationships maintained by the agents in a system, and the procedures by which the relationships are maintained, we can allow for significant selectivity in monitoring, while providing effective monitoring capabilities. We call such monitoring using knowledge of social relationships *socially-attentive monitoring*, to differentiate it from other types of monitoring which rely on other types of knowledge, such as knowledge of the goals of the agents' tasks, or knowledge of the duration of steps that they have to take. Here, the term social relationship is used to

denote a relation on attributes of multiple agents' states. Socially-attentive monitoring in the convoy example involves verifying that agents have common destination and heading, that their beliefs in driving as a convoy are mutual, etc. For instance, if the agents are observed to head in different directions, they clearly do not have a common heading. This is different than monitoring whether their chosen (common) heading leads towards their (agreed upon) destination.

We make three key assumptions in the techniques we present. First, in defining a relationship failure, we make the assumption that the team does not allow an agent to temporarily violate the team relationships in service of other commitments which are not shared by the team. For instance, we assume an agent should not (under non-failure conditions) temporarily suspend execution of a task it is carrying out for one team, in service of carrying out a task for a different team.

We make a second key assumption in utilizing plan-recognition for monitoring. We assume that the plan-library used for recognizing the other agents in complete and correct. This is a common assumption in plan-recognition literature (e.g., Kautz & Allen, 1986), and is often true in collaborative settings. It does not imply that the agents are homogeneous, but that the range of plans agents utilize is recognizable to the monitoring agent.

Finally, a related assumption that we make is that the observations of the monitoring agent are correct, involving no failures in monitored actions and in our observation. In other words, we assume that if an agent is observed to have taken an action, than this action was intended and executed by the monitored agent. For example, if a helicopter is observed as landing, we have assumed that indeed the landing action was intended and carried out, and that our sensing of the action was correct.

1.1 Monitoring Selectivity in Relationship Failure Detection

In Part I of this dissertation, we focus our investigation on the task of detecting failures in the social relationships that ideally hold between agents in a monitored team. Such monitoring is a critical task, as failures to maintain the team's relationships

can often lead to catastrophic failures on the part of the team, lack of cooperative behavior and lack of coordination. Such failures are often the result of individual agent failures, such as failures in an agent's sensors and actuators. Thus relationship failures cover a large class of failures, and their detection promotes robust individual operation.

We explore socially-attentive algorithms for detecting teamwork failures under various conditions of uncertainty, resulting from the necessity of selectivity. We analytically show that despite the presence of uncertainty about the actual state of monitored agents, a centralized *active* monitoring scheme can guarantee failure detection that is either sound and incomplete, or complete and unsound. However, this requires monitoring all agents in a team, and reasoning about multiple hypotheses as to their actual state. We then show that active distributed teamwork monitoring results in sound and complete detection capabilities, despite using a much simpler algorithm. By exploiting the agents' local states, which are not available to the centralized algorithm, the distributed algorithm: (a) uses only a single, possibly incorrect hypothesis of the actual state of monitored agents, and (b) involves monitoring only key agents in a team, not necessarily all team-members (thus allowing even greater selectivity). Using a transformation on the analytical constructs, we show analogous results for centralized failure-detection in mutual-exclusion coordination relationships.

We also conduct an empirical investigation of socially-attentive monitoring in teams. We present an implemented general socially-attentive monitoring framework in which the expected ideal social relationships that are to be maintained by the agents are compared to the actual social relationships. Discrepancies are detected as possible failures and diagnosed. We apply this framework in two different complex, dynamic, multi-agent domains, in service of monitoring various social relationships, both on-line and off-line. Both of these domains involve multiple interacting agents in collaborative and adversarial settings, with uncertainties in both perception and action. The empirical results for active monitoring support our analytical results.

In investigating off-line teamwork failure monitoring, we demonstrate a re-use of the failure-detection techniques, in service of a quantitative measure which measures some aspects of teamwork quality. This measure fairs better than the general task-performance measures in measuring teamwork-related questions. It is significantly

more sensitive, requiring much less data to be useful, and providing more fine-grained information.

1.2 Monitoring Selectivity in Monitoring Distributed Teams

In Part II, we investigate the monitoring selectivity problem in a second monitoring task, which involves identifying the state of a distributed team. Recent years are seeing tremendous growth of applications involving complex, distributed, multi-agent organizations (systems), which involve many heterogeneous software agents, built by different designers with different goals and capabilities in mind (Huhns & Singh, 1998).

The monitoring selectivity problem here is more difficult than in the failure-detection task. First, the distribution of the team leads to limited observability of agents' actions, and thus significant selectivity in monitoring is imposed on the monitoring agent a-priori, leading to an explosion of uncertainty about the state of monitored agents. For instance, information agents may display information to a human user on its workstation screen, but such action is difficult to observe from a monitoring system hundreds of miles away, and must be hypothesized by the monitoring agent. A second factor in the difficulty of the monitoring selectivity problem in this task is that the monitoring system is required to accurately identify the state of the monitored team in spite of the significant uncertainty, i.e., to prove effective, the monitoring system cannot resort to identifying just enough of the state to detect a failure.

We apply socially-attentive monitoring techniques to provide answers to the monitoring selectivity problem, tackling the significant uncertainty that results from the selectivity imposed by the problem constraints. First, we present a number of socially-attentive monitoring techniques which overcome much of the uncertainty in distributed team monitoring by utilizing explicit knowledge of the relationships in the monitored team, and the procedures employed by the team to maintain those relationships. Then, we explore a number of algorithms for limited-observations

monitoring which provide a trade-off between computational requirements and expressivity. We show that a fully expressive algorithm that can reason explicitly about failure hypotheses has space requirements linear in the number of agents, and maintains a possibly exponential (in the number of monitored agents) hypotheses. In contrast, a reduced-expressivity algorithm can only detect some failures (but not reason explicitly about them), but has space requirements that are constant in the number of monitored agents, and maintains only a constant (with respect to the number of agents) number of hypotheses.

1.3 Organization of This Thesis

This dissertation's organization is presented in Figure 1.1. It is organized in two parts. Part I discusses answers to the monitoring selectivity problem in failure-detection and diagnosis: Chapter 2 presents motivating examples and background. Chapter 3 presents the socially-attentive failure-detection framework; Chapter 4 explores monitoring selectivity in centralized teamwork monitoring; and Chapter 5 explores monitoring selectivity in distributed teamwork monitoring. Chapter 6 demonstrates the generality of our failure-detection framework (presented in Chapter 3) by re-using it in an off-line quantitative teamwork analysis implementation. Chapter 7 discusses monitoring selectivity in additional relationship failure-detection (in centralized configuration).

Part II investigates monitoring of distributed teams, based on their communications. Chapter 8 presents background and motivating examples. Chapter 9 discusses socially-attentive monitoring techniques for accurate monitoring. Chapter 10 discusses methods for scaling up monitoring and improving efficiency, at the cost of expressivity.

Finally, Chapter 11 presents related work, and Chapter 12 concludes. The appendices provide additional information. Appendix A lists relevant publications. Appendix B fleshes out the proofs for theorems presented. Appendix C describes the probabilistic plan recognition representation and algorithms (co-developed with David V. Pynadath) used as the basis for our investigation in Part II. Appendix D contains pseudo-code for the socially-attentive failure-detection algorithms.

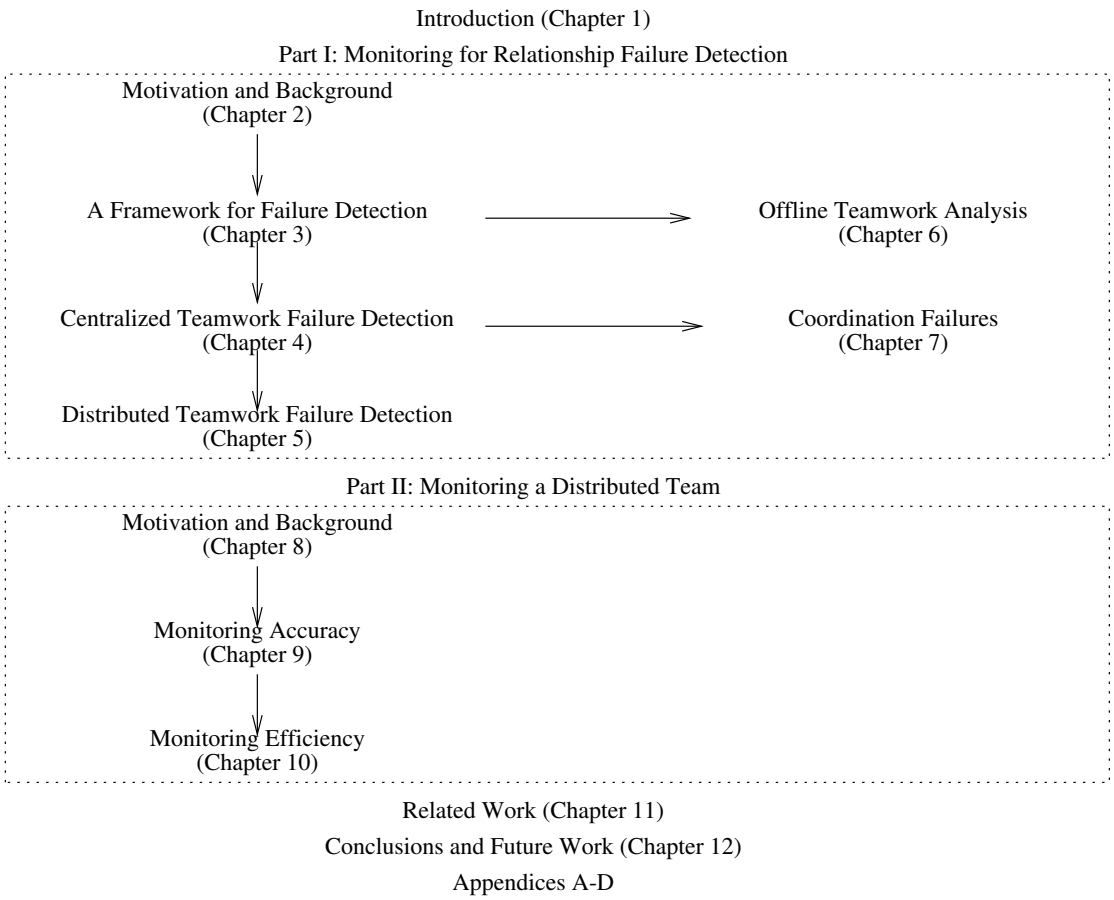


Figure 1.1: Organization of this Dissertation.

Part I

Monitoring for Relationship Failure Detection

Chapter 2

Motivation and Background: Failure Detection

The motivation for our focus on monitoring in service of relationship failure detection stems from our growing frustration with the significant software maintenance efforts in two of our application domains. In the ModSAF domain, a high-fidelity battlefield virtual environment (Calder et al., 1993), we have been involved in the development of synthetic helicopter pilots (Tambe et al., 1995). In the RoboCup soccer simulation domain (Kitano et al., 1997) we have been involved in developing synthetic soccer players (Marsella, Adibi, Al-Onaizan, Kaminka, Muslea, Tallis, & Tambe, 1999). The environments in both domains are dynamic and complex, and have many uncertainties: the behavior of other agents (some adversarial, some cooperative), unreliable communications and sensors, actions which may not execute as intended, etc. Agents in these environments are therefore presented with countless opportunities for failure despite the designers' best efforts.

Some examples may serve to illustrate. The following two examples are actual failures that occurred in the ModSAF domain. We will use these two to illustrate and explore socially-attentive monitoring throughout this part:

Example 1. Here, a team of three helicopter pilot agents were to fly to a specified way-point (a given position), where one of the team-members, the *scout*, was to fly forward towards the enemy, while its teammates (*attackers*) land and wait for its signal. All of the agents monitored for the way-point. However, due to an unexpected sensor failure, one of the attackers failed to sense the way-point. So while the other attacker correctly landed, the failing attacker continued to fly forward with the scout (see Figure 2.1 for a screen shot illustrating this failure).

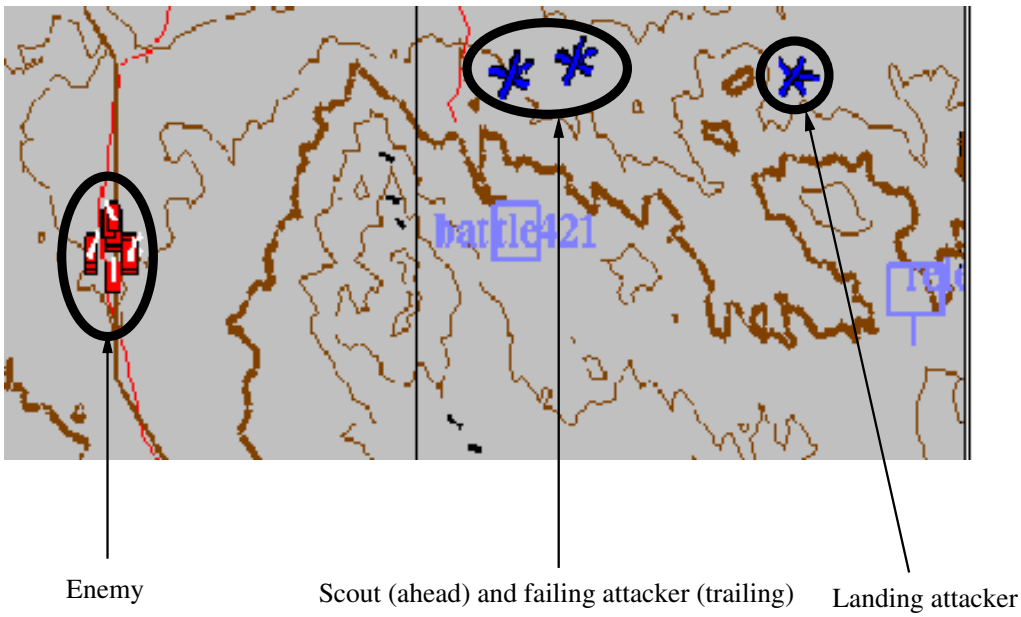


Figure 2.1: A plan-view display (the ModSAF domain) illustrating the failure in Example1. The thick wavy lines are contour lines.

Example 2. In a different run, after all three agents reached the way-point and detected it, the scout has gone forward and identified the enemy. It then sent a message to the waiting attackers to join it and attack the enemy. One of the attackers did not receive the message, and so it remained behind indefinitely while the scout and the other attacker continued the mission alone.

We have collected dozens of similar reports in both the ModSAF and RoboCup domain. In general, such failures are difficult to anticipate in design time, due to the huge number of possible states. The agents therefore easily find themselves in novel states which have not been foreseen by the developer, and the monitoring conditions and communications in place proved insufficient: In none of the failure cases reported did the agents involved detect, let alone correct, their erroneous behavior. Each agent believed the other agents to be acting in coordination with it, since no communication was received from the other agents to indicate otherwise. However, the agents were violating the collaboration relationships between them, as the agents came to disagree on what plan is being executed—a collaboration relationship failure had occurred. Preliminary empirical results show that upwards of 40% of failures reported involved relationship violations (relationship failures).

Human observers, however, were typically quick to notice these failures, because of the clear social misbehavior of the agents in these cases. They were able to infer that a failure has occurred despite not knowing what exactly happened. For instance, seeing an attacker continuing to fly ahead despite its teammates' switching to a different plan (which the human observers inferred from the fact that one of the teammates, the other attacker, has landed) is sufficient for an observer to detect that something has gone amiss—without knowing what the different plan was.

The monitoring selectivity problem emerges in these examples. On one hand, our analysis showed that the agents were not monitoring each other sufficiently. However, a naive solution of continuous communications between the agents was clearly impractical since: (i) the agents are operating in a hostile environment; (ii) the communications overheads would have been prohibitive; and (iii) in fact, it was the communications equipment itself that broke down in some cases. We therefore sought practical ways to achieve quick detection of failure, based on the limited ambiguous knowledge that was available to a monitoring agent.

Chapter 3

Socially-Attentive Monitoring for Failure Detection

We present an overview of the general structure of a socially-attentive monitoring system, shown in Figure 3.1. It consists of the following components: (1) a social relationship knowledge-base containing models of the relationships that should hold among the monitored agents, enabling generation of *expected ideal behavior* in terms of relationships (Section 3.1); (2) an agent and team modeling component, responsible for collecting and representing knowledge about the monitored agents' *actual behavior* (Section 3.2); (3) a relationship failure-detection component that monitors for violations of relationships among monitored agents by contrasting the expected and actual behavior (Section 3.3); and (4) a relationship diagnosis component that verifies the failures, and provides an explanation for them (Section 3.4). The resulting explanation (diagnosis) is then used for recovery, e.g., by a negotiations system (Kraus, Sycara, & Evenchik, 1998), or a general (re)planner (Ambros-Ingerson & Steel, 1988).

3.1 A Knowledge-Base of Relationship Models

We take a relationship among agents to be a relation on their state attributes. A relationship model thus specifies how different attributes of an agent's state are related to those of other agents in a multi-agent system. These attributes can include the beliefs held by the agents, their goals, plans, actions, etc. For example, many teamwork relationship models require that team-members have mutual belief in a joint goal (Cohen & Levesque, 1991; Jennings, 1995). A spatial formation relationship (Parker, 1993; Balch, 1998) specifies relative distances, and velocities that are

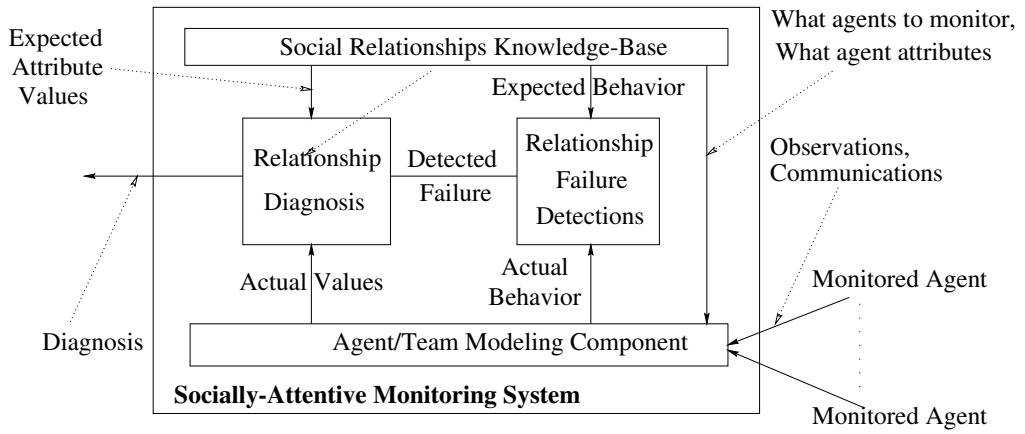


Figure 3.1: The general structure of a socially-attentive monitoring system.

to be maintained by a group of agents (in our domain, helicopter pilots). Coordination relationships may specify temporal relationships that are to hold among the actions of two agents, e.g., business contractors (Malone & Crowston, 1991). All such relationships are *social*—they explicitly specify how multiple agents are to act and what they are to believe if they are to maintain the relationships between them.

The relationship knowledge-base contains models of the relationships that are supposed to hold in the system, and specifies the agents that are participating in the relationships. The knowledge-base guides the agent-modeling component in selecting agents to be monitored, and what attributes of their state need be represented (for detection and diagnosis). It is used by the failure detection component to generate expectations which are contrasted with actual relationships maintained by the agents. And it provides the diagnosis component with detailed information about how agents' states' attributes are related, to drive the diagnosis process. Our implementation of socially-attentive monitoring in teams uses four types of relationships: *formations*, *role-similarity*, *mutual exclusion*, and *teamwork*.

For teamwork monitoring we use the STEAM (Tambe, 1997) general domain-independent model of teamwork, which is based on Cohen and Levesque's Joint Intentions Framework (Levesque, Cohen, & Nunes, 1990; Cohen & Levesque, 1991) and Grosz, Sidner, and Kraus's SharedPlans (Grosz & Sidner, 1990; Grosz & Kraus, 1996, 1999). However, other teamwork models (e.g., Jennings, 1995; Rich & Sidner, 1997) may be used instead of STEAM. Although STEAM is used by our pilot and soccer agents to generate collaborative behavior, it is reused here independently

in service of monitoring, i.e., monitored agents are assumed to be a team, and STEAM is used in monitoring their teamwork. STEAM require mutual belief by team-members in certain beliefs that the agents may hold, and agreement on a joint goals and plans. These requirements are monitored in our system for violations which indicate failure (see Section 3.3 and Section 3.4 for details of how STEAM is used). The other relationship models are used only in a secondary monitoring role. They will be discussed in greater length in Chapter 7.

3.2 Knowledge of Monitored Agents and Team

The agent modeling component is responsible for acquiring and maintaining knowledge about monitored agents. This knowledge is used to construct the *actual* relations that exist between agents' states' attributes, which are compared to the *ideal expected* relations. In this section, we describe the plan recognition capabilities of the agent-modeling component in our implementation and experiments, i.e., the extent of the knowledge that could be maintained about monitored agents' plans if necessary. Later sections show that in fact limited, possibly inaccurate, knowledge is sufficient for effective failure detection. Thus implementations may use optimized agent-modeling algorithms rather than these full capabilities. Section 3.4 will discuss additional agent-modeling capabilities, necessary for diagnosis.

3.2.1 Representation

For monitoring teamwork relationships, we have found that representing agents in terms of their selected hierarchical reactive plans enables quick monitoring of their state, and also facilitates further inference of the monitored agents' beliefs, goals, and unobservable actions, since they capture the agents' decision processes.

In this representation, reactive plans (Firby, 1987; Newell, 1990) form a single decomposition hierarchy (a tree) that represents the alternative controlling processes of each agent. The root of the hierarchy represents the overall high-level process that the agent executes. Each node in the hierarchy is a reactive plan (hereafter referred to simply as a plan), which may require sub-plans to be executed as steps to achieve the plan's goals.

Each plan has selection conditions (also referred to as preconditions) for when it is applicable. Such conditions can refer to sensor readings or the internal state of the agent. When these conditions are fulfilled, the plan is selected for execution. When more then one plan is applicable, search-control rules are used to decide on a single plan that will be executed. Sub-plans can test (as part of their preconditions) for the selection of a parent plan in service of which they are executing. Thus at each given moment, the agent is executing a single path (root to a leaf) through the hierarchy. This path is composed of plans at different levels, one selected plan per level. Each plan also has termination conditions, which can be used to terminate the execution of plan, and de-select it. When a plan is terminated, its current sub-plans (if any) are terminated as well. Achievement of the termination conditions of the root, therefore, signify completion of the agent's task (either successfully, or unsuccessfully).

Figure 3.2 presents a small portion of such a hierarchy, created for the ModSAF domain. In the case of Example 1, prior to the way-point, *each of the agents* was executing the path beginning with `execute-mission` as highest-level plan, through `fly-flight-plan`, `fly-route`, `traveling` and `low-level` (see also left hierarchy in Figure 3.4). Upon reaching the way-point, sensor readings were supposed to cause a termination condition of `fly-flight-plan` to become true, thus terminating `fly-flight-plan` for each helicopter. Then, the same sensor reading would be used as a precondition for the `wait-at-point` plan, causing it to be selected. Thus all helicopters were supposed to switch from `fly-flight-plan` and its descendents to `wait-at-point`. The attackers would then select `just-wait` as a child of `wait-at-point`, while the scout would select `scout-forward` and its descendents. Of course, the failing attacker did not detect the way-point and so the termination conditions for `fly-flight-plan` and the selection conditions for `wait-at-point` were not satisfied and the failing attacker continued to execute `fly-flight-plan` and its descendents.

3.2.2 Acquisition

From a practical perspective, while the agents may cooperatively report to the monitoring agent on their own state using communications, it requires communication

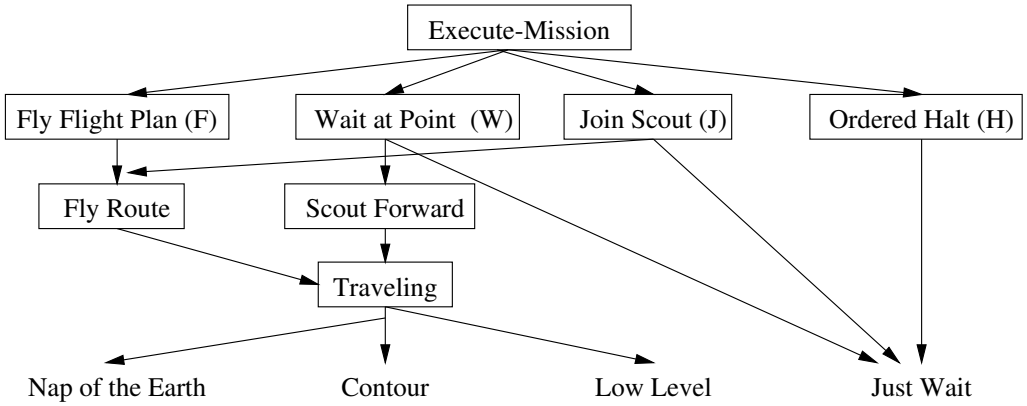


Figure 3.2: Portion of Hierarchical Reactive Plan Library for ModSAF Domain (Team plans are boxed. These are explained in Section 3.3).

channels to be sufficiently fast, reliable and secure. This is unfortunately not possible in many realistic domains, as our examples demonstrate (Chapter 2).

Alternatively, a monitor may use plan recognition to infer the agents' unobservable state from their observable behavior. This approach is unintrusive and robust in face of communication failures. Of course, the monitor may still benefit from focused communications with the other agents, but would not be critically dependent on them.

To enable plan recognition using reactive plans (our chosen representation), we have employed a reactive plan recognition algorithm called RESL (REal-time Situated Least-commitments). The key capability required is to allow explicit maintenance of hierarchical plan hypotheses matching each agent's observed behavior, while pruning of hypotheses which are deemed incorrect or useless for monitoring purposes. RESL works by expanding the entire plan-library hierarchy for each modeled agent, and tagging all paths matching the observed behavior of the agent being modeled (see Appendix D for pseudo-code for the algorithm). Heuristics and external knowledge may be used to eliminate paths (hypotheses) which are deemed inappropriate—indeed such heuristics will be explored shortly. RESL's basic approach is very similar to previous work in reactive plan recognition (Rao, 1994) and team-tracking (Tambe, 1996), which have been used successfully in the ModSAF domain, and share many of RESL's properties. However, RESL adds belief-inference capabilities which are used in the diagnosis process, discussed below (Section 3.4).

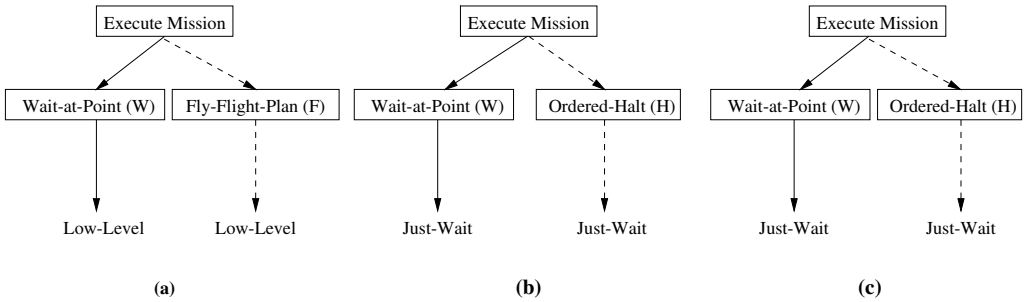


Figure 3.3: Scout (a) and Attackers' (b, c) actual and recognized *abbreviated* reactive plan hierarchies.

Figure 3.3 gives a simplified presentation of the plan hierarchies for a variation of Example1, in which all the agents correctly detected the way-point, i.e., no failure has occurred (note that some plans at intermediate levels have been abstracted out in the figure). The scout (Figure 3.3a) and the two attackers (Figures 3.3b, 3.3c) switched from the `fly-flight-plan` plan (denoted by `F)` to the `wait-at-point` plan (denoted by `W`). An outside observer using RESL infers explanations for each agent's behavior by observing the agents. The scout continues to fly ahead, its speed and altitude matching `low-level`, one of the possible flight-methods under both the `fly-flight-plan` (`F`) and `wait-at-point` (`W`) plans. Thus they are both tagged as possible hypotheses for the scout's executing plan hierarchy. Similarly, as the attackers land, RESL recognizes that they are executing the `just-wait` plan. However, this plan can be used in service of either the `W` or the `ordered-halt` (`H`) plan—a plan in which the helicopters are ordered by their headquarters to land immediately. Thus both `H` and `W` are tagged as explanations for each of the attackers' states (at the second level of the hierarchies). For all agents, RESL identifies the plan `execute-mission` as the top-level plan. In this illustration, the actual executing paths of the agents are marked with filled arrows. Other *individual modeling* hypotheses that match the observed behavior are marked using dashed arrows. An outside observer, of course, has no way of knowing which of the possible hypotheses is correct.

Once individual modeling hypotheses are acquired for each individual agent (using plan recognition in our implementation, but potentially also by communications), the monitoring agent must combine them to create team-modeling hypotheses as to

the state of the team as a whole. The monitoring agent selects a single individual modeling hypothesis for each individual agent and combines them into a single team-modeling hypothesis. Several such team-modeling hypotheses are possible given multiple hypotheses for individual agents. For instance, in Figure 3.3, while all team-hypotheses will have `execution-mission` as the top-level plan, there are eight different team-hypotheses which can be differentiated by their second-level plan: (W,W,W), (W,W,H), (W,H,W), (W,H,H), (F,W,W), (F,W,H), (F,H,W), (F,H,H). If the observer is a member of the team, it knows what it is executing itself, but would still have multiple hypotheses about its teammates' states. For instance, if the attacker in Figure 3.3b is monitoring its teammates, its hypotheses at the second level would be (W,W,W), (W,W,H), (F,W,W), (F,W,H).

To avoid explicitly representing a combinatorial number of hypotheses, RESL explicitly maintains all candidate hypotheses for each agent individually, but not all combinations of individual models as team hypotheses. Instead, these combinations are implicitly represented. Thus the number of hypotheses explicitly maintained grows linearly in the number of agents.

3.3 Relationship Violation Detection

The failure-detection component detects violations of the social relationships that should hold among agents. This is done by comparing the ideal expected relationships to their actual maintenance by the agents. In our teamwork monitoring implementation, which uses STEAM (Tambe, 1997), the relationship model requires team-members to always agree on which *team-plan* is jointly executed by the team, similarly to Joint Responsibility (Jennings, 1995), and SharedPlans (Grosz & Kraus, 1996). If this requirement fails in actuality (i.e., the agents are executing different team plans) then a teamwork failure has occurred. In its current form, STEAM does not allow (under non-failure conditions) an individual agent to temporarily suspend execution of team plans in service of other commitments which are not shared by the team. For instance, we assume an agent should not temporarily suspend execution of one team plan so that it can execute a different team plan, as member of a different team. However, the agent may choose to do so responsibly, for instance by informing its teammates of its decision.

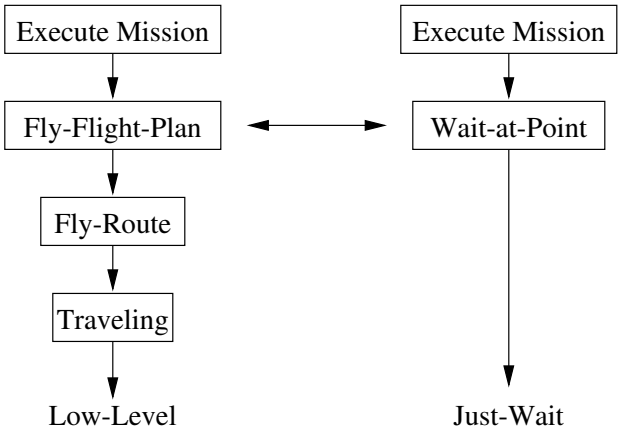


Figure 3.4: Comparing two hierarchical plans. The top-most difference is at level 2.

The basic teamwork failure detection algorithm is as follows. The monitored agents' plan hierarchies are processed in a top-down manner. The detection component uses the teamwork model to tag designer-specified plans as team plans, explicitly representing joint activity by the team (these plans are boxed in Figures 3.2, 3.4 and 3.3). The team-plans in equal depths of the hierarchies are used to create team-modeling hypotheses. For each hypothesis, the plans of different agents are compared to detect disagreements. Any difference found is an indication of failure. If no differences are found, or if the comparison reaches individual plans (non-team, therefore non-boxed in the figures) no failure is detected. Individual plans, which may be chosen by an agent individually in service of team plans are not boxed in these figures, and are handled using other relationships as discussed in Chapter 7.

For instance, suppose the failing attacker from Example1 is monitoring the other attacker. Figure 3.4 shows its view of its own hierarchical plan on the left. The path on the right represents the state of the other attacker (who has landed). This state has been inferred in this example from observations made by the monitoring attacker (here, we are assuming that the plan recognition process has resulted in one correct hypothesis for each agent. We will discuss more realistic settings below). In Figure 3.4, the difference that would be detected is marked by the arrow between the two plans at the second level from the top. While the failing attacker is executing the `fly-flight-plan` team-plan (on the left), the other attacker is executing the `wait-at-point` team-plan (on the right). The disagreement on which team-plan is to be executed is a failure of teamwork.

Thus the monitoring selectivity problem is raised. The lack of perfect knowledge of team-members' state (only some actions are observed) leads to uncertainty about the team's state, expressed by multiple team-modeling hypotheses (Section 3.2). However, detecting disagreements is difficult with multiple team-modeling hypotheses, since they may imply contradictory results with respect to failure detection: Some hypotheses may imply that a failure had occurred in the team, while others may not. Unfortunately, this is to be expected in realistic applications. For instance, Figure 3.3 (Section 3.2) shows several hypotheses that are possible based on the same observations. However, one of the hypotheses, (`W`,`W`,`W`), implies no failure has occurred—all the agents are in agreement on which team-plan is executing—while another hypothesis, (`F`,`W`,`H`), implies failures have occurred.

To limit reasoning to only a small number of team hypotheses, while not restricting failure-detection capabilities, we use a disambiguation heuristic that ranks team-modeling hypotheses by the level of *coherence* they represent. This heuristic is provided as an initial solution. Later sections will examine additional heuristics.

Definition 1. The *coherence* level of a multi-agent modeling hypothesis is defined as the ratio of the number of agents modeled to the number of different plans contained in the hypothesis:

$$\frac{\#\,of\ Agents}{\#\,of\ Different\ Plans}$$

This definition results in a partial ordering of the hypotheses set, from the least coherent hypothesis (one that assigns each agent a different plan than its teammates), to the most coherent hypothesis (that assigns the same plan to all teammembers). For instance, the hypothesis (`F`,`W`,`H`) would have the lowest level of coherence, 1, since it implies complete breakdown of teamwork—every agent is executing a different plan. The hypothesis (`W`,`W`,`W`) would have a coherence level of 3, the highest level of coherence for the group of three agents, since they are all assigned the same plan. Ranked between them would be the hypothesis (`W`,`W`,`H`), with a single teamwork failure (disagreement on W and H) and a coherence level of 3/2.

The detection component selects a single maximally-coherent team-modeling hypothesis (ties broken randomly). The intuition for using coherence is that failures to agree occur despite the agents' attempts at teamwork. Thus we expect

more agreements than disagreements in the team. The coherence level of a team-hypothesis is inversely related to the number of teamwork failures implied by the hypothesis. Selecting a maximally-coherent hypothesis therefore corresponds to the minimum-number-of-failures heuristic commonly used in diagnosis (Hamscher, Console, & de Kleer, 1992).

For the case depicted in Figure 3.3, the complete detection process may be conceptualized as follows[1]. Suppose that one of the attackers, whose hierarchy is described in Figure 3.3b, is monitoring the team. First, it collects the plan hypotheses at the top of the hierarchy for each agent (including itself). In this case, they are {`execute-mission`}, {`execute-mission`}, {`execute-mission`}. Only one team-modeling hypothesis can be built from these: (`execute-mission`, `execute-mission`, `execute-mission`). Since this hypothesis shows no disagreement occurs at this level, the process continues to the second level. Here, the hypotheses for the first agent on the left are {`F`,`W`}, for the monitoring second agent (since it knows its own state) there is only one possibility {`W`}, and for the third agent {`W`,`H`}. As we saw above, the maximally team-coherent hypothesis is (`W`,`W`,`W`) which is selected. Since it does not indicate failure, the process continues to the third level. Here the agents are executing individual plans, and so the comparison process stops. Algorithm D.2 in Appendix D provides greater details about this process.

When sub-teams are introduced, a difference between team-plans may be explained by the agents in question being a part of different sub-teams. Sub-team-members still have to agree between themselves on the joint sub-team plans, but these may differ from one sub-team to the next. For now, let us assume that the teams under consideration are *simple teams*, as defined in Definition 2. We make this definition in service of later analytical results in which it will appear as a condition. We return to the issue of sub-teams in Section 7.1.

Definition 2. We say that a team T is *simple* if its plan hierarchy involves no different team plans which are to be executed by different sub-teams.

Intuitively, the idea is that in a simple team, all members of the team jointly execute each of the team plans in the hierarchy. This definition is somewhat similar

[1] Other implementations may use optimized algorithms in which the heuristics are integrated into the agent-modeling algorithm. See Chapter 10 for one such algorithm.

to the definition of a *ground team* in (Kinny, Ljungberg, Rao, Sonenberg, Tidhar, & Werner, 1992), but it does not allow sub-team-members of a team to have a joint plan which is different than that of other members.

3.4 Relationship Diagnosis

The diagnosis component constructs an explanation for the detected failure, identifying the failure state and facilitating recovery. The diagnosis is given in terms of a set of agent belief differences (inconsistencies) that explains the failure to maintain the relationship. The starting point for this process is the detected failure (e.g., the difference in team-plans). The diagnosis process then compares the beliefs of the agents involved to produce a set of inconsistent beliefs that explain the failure.

Two problems exist in practical applications of this procedure. First, the monitoring agent is not likely to have access to all of the beliefs held by the monitored agents, since it is not feasible in practice to communicate all the agents' beliefs to each other. Second, each agent in a real-world domain may have many beliefs, and many of them will vary among the agents, though most of them will be irrelevant to the diagnosis. Thus relevant knowledge may simply not be accessible, or may be hidden in mountains of irrelevant facts.

To gain knowledge of the beliefs of monitored agents without relying on communications, the diagnosis process uses a process of belief ascription. The agent-modeling component (using RESL in our implementation) maintains knowledge about the selection and termination conditions of recognized plans (hypotheses). For each recognized plan hypothesis, the modeling component infers that any termination conditions for the plan are believed to be false by the monitored agent (since it has not terminated the plan). We have also found it useful to use an additional heuristic, and infer that the selection conditions (preconditions) for any plan which *has just begun execution* are true. The idea is that when a plan is selected for execution, its preconditions are likely to hold, at least for a short period of time. This heuristic involves an explicit assumption on the part of our system that the new plan is recognized as soon as it begins execution. Designers in other domains will need to verify that this assumption holds.

For each agent i, the inferred termination and selection conditions make up a set of beliefs B_i for the agent. For instance, suppose an agent is hypothesized to have just switched from executing `fly-flight-plan` to `wait-at-point`. RESL infers that the agent believes that the way-point was just detected (a selection condition for `wait-at-point`). In addition, RESL infers that the agent believes that an enemy was not seen, and that no order was received from base to halt the mission (negated termination conditions of `wait-at-point`).

To determine the facts that are relevant to the failure, the diagnosis component uses the teamwork model (in our implementation, STEAM (Tambe, 1997)). The teamwork model dictates which beliefs the agents hold must be mutually believed by all the agents in the team. Any difference that is detected in those beliefs is a certain failure, as the team-members do not agree on issues on which agreement is mandatory to participation in the team. The teamwork model thus specifies that the beliefs contained in the B_i sets should be mutual, and should therefore be consistent:

$$\bigcup_i B_i \nvdash \bot$$

If an inconsistency is detected, the diagnosis procedure looks for contradictions (disagreements) that would cause the difference in team-plan selection. A difference in beliefs serves as the diagnosis, allowing the monitoring agent to initiate a process of recovery (e.g., by negotiating about the conflicting beliefs (Kraus et al., 1998)).

For example, as shown in Section 3.3, the two attackers in Example 1 (Chapter 2) differ in their choice of a team-plan: One attacker is continuing execution of the `fly-flight-plan` plan, in which the helicopters fly in formation. The other attacker has detected the way-point, terminated `fly-flight-plan` and has switched to `wait-at-point`, landing immediately (Figure 3.4). When the failing attacker monitors its teammate, it detects a difference in the team-plans (Section 3.3), and the detected difference is passed to diagnosis. The failing attacker makes the following inferences:

1. `Fly-flight-plan` has three termination conditions: (a) detecting the way-point, (b) seeing the enemy, or (c) receiving an order to halt. The failing

attacker (left hierarchy in Figure 3.4) knows its own belief that none of these conditions hold, and thus

$$B_1 = \{\neg WayPoint, \neg Enemy, \neg HaltOrder\}$$

2. `Wait-at-point` has one selection condition: the way-point has been detected. Its termination condition is that the scout has sent a message to join it, having identified the enemy's position. The diagnosis component in this case therefore infers that for the other attacker (right hierarchy in Figure 3.4)

$$B_2 = \{WayPoint, \neg ScoutMessageReceived\}$$

Combining B_1 and B_2, we get

$$\begin{aligned} B_1 \cup B_2 \quad = \quad & \{\neg WayPoint, WayPoint, \neg Enemy, \\ & \neg ScoutMessageReceived, \neg HaltOrder\} \end{aligned}$$

which is inconsistent. The inconsistency (disagreement between the attackers) is $\{\neg WayPoint, WayPoint\}$, i.e., contradictory beliefs about $Waypoint$. Thus now the failing attacker knows that its team-mate has seen the way-point. It can choose to quietly adapt this belief, thereby terminating its own `fly-flight-plan` and selecting `wait-at-point`, or it may choose other recovery actions, such as negotiating with the other attacker on whether the way-point has been reached.

We have found these diagnosis procedures to be useful in many of the failures detected by socially-attentive monitoring (see Chapter 4 for evaluation and discussion). However, since this dissertation focuses on the monitoring selectivity problem in *detection*, we leave further investigation of the diagnosis procedures to future work.

Chapter 4

Monitoring Selectivity in Centralized Teamwork Monitoring

Using the socially-attentive framework of Chapter 3 we systematically examine all failure permutations of Examples 1 and 2 (Chapter 2) under a centralized teamwork monitoring configuration, where a single team-member is monitoring the team. We vary the agents failing (attacker, attacker and scout, etc.) and the role of the monitoring agent (attacker or scout). We report on the empirical results of detecting and diagnosing failures in all cases. Using these empirical results as a guide, we explore centralized teamwork monitoring analytically. We show that even under monitoring uncertainty, centralized teamwork monitoring can provide either sound or complete detection results (but not both).

As a starting point for our exploration, the monitoring agent uses a single maximally-coherent team-modeling hypothesis as discussed in Section 3.3. We begin with Example 2. The normal order of execution is `wait-at-point` (`W`), followed by `join-scout` (`J`). During the execution of `W`, the two attackers land and wait for the scout to visually identify the enemy's position. Upon identification, the scout sends them a message to join it, which triggers the selection of the `J` plan, and the termination of the `W` plan. When executing `J`, the scout hovers at low altitude, waiting for the attackers to join it. Any failures here are on the part of the attackers (they cannot receive the message) or on the part of the scout (it cannot send it). These failures arise, for instance, if the radio is broken or team-members are out of range. When an agent fails, it continues to execute `W` instead of switching to `J`.

Case #	Actual Executing Plans			Relationship Failure Occurred?	Physical Failure
	Attacker A1	Attacker A2	Scout A3		
1	J	J	J	-	-
2	W	J	J	+	A1 fails to receive
3	J	W	J	+	A2 fails to receive
4	W	W	J	+	A3's message lost
5	W	W	W	-	Enemy not identified

Table 4.1: All possible failure permutations of the broken radio-link scenario (Example 2).

Table 4.1 summarizes the permutations of Example 2. The permutation number appears in the first column. The next three columns show the actual plans selected by the three agents A1, A2 and A3 in each permutation. The second-to-last column shows whether a relationship failure has occurred in each case, i.e., whether disagreement exists between the agents. Finally, the last column details the physical conditions in each case. There are five possible failure permutations: In case 1, none of the agents failed. In cases 2 and 3 one attacker failed. In case 4 the scout failed to send a message or both attackers failed to receive it. In case 5 the scout does not identify the enemy's position (so no message is sent, and all three agents continue to execute the `W` plan). Other permutations are not possible, since no attacker can switch to the `J` plan without the scout.

For instance, case 2 in Table 4.1 corresponds to Example 2. The scout (A3) has detected the enemy, switched to plan `J`, and sent a message to the attackers to join it. One attacker (A2) received the message, switched to plan `J`, and began flying towards the scout. However, the remaining attacker (A1) failed to receive the message, and so it maintains its position, continuing to execute `W` and failing to switch to `J`. Since the agents are no longer in agreement on which team plan should be jointly executed, a teamwork failure has occurred. Condition monitors were used in the original failure case to monitor for the scout's message. However failures in communications resulted in these monitoring conditions to be rendered useless.

One key issue is raised by case 5 in Table 4.1. Here, due to the scout's inability to identify the enemy's position (perhaps due to failure on the scout's part, perhaps

Case #	A3's Hypothesized Executing Plans			Relationship Failure Detected?	Diagnosis Success?	Detection Class
	Attacker A1	Attacker A2	Scout A3			
1	J	J	J	-	n/a	True Negative
2	W	J	J	+	+	True Positive
3	J	W	J	+	+	True Positive
4	H	H	J	+	-	True Positive
5	W	W	W	-	n/a	True Negative

Table 4.2: Scout's (A3) monitoring results in all permutations of Example 2.

because the enemy is simply not there), the three helicopter pilots remain in agreement that the enemy has not been identified. Here, even though clearly the pilots are failing to make progress towards the task goals (the scout continues to search for the enemy indefinitely), no relationship failure is taking place, since the agents are maintaining the teamwork relationship while failing to make progress. This clearly demonstrates that not all failures are necessarily relationship failures.

Table 4.2 presents the results of the scout monitoring its teammates in Example 2, using a maximally team-coherent hypothesis as the basis for detection. The first column again shows the case number, for reference into Table 4.1. The next three columns show the scout's (A3's) hypothesis about what plan each agent is executing according to the maximal coherence heuristic. The next two columns show whether a failure was detected, and whether it was diagnosed correctly. The last column shows the detection class (discussed below).

For example, case 2 in Table 4.2 shows the results of the scout monitoring in the original failure in Example 2 (Chapter 2). Using RESL, and selecting a maximally-coherent hypothesis, the scout hypothesizes that the non-moving attacker is executing `W` (case 2, column 2), while the moving attacker is executing `J` (case 2, column 3). The scout of course knows that its own selected plan `J` (case 2, column 4). A violation of the teamwork relationship is thus detected (case 2, column 5), since A1's `W` is not in agreement with the rest of the team's `J`. Furthermore, the diagnosis was successful in identifying the cause for the failure, i.e., the fact that the enemy's position has been identified by the scout, but no knowledge of this was passed on to the failing attacker (case 2, column 6).

The last column of Table 4.2 shows the detection class of each failure. The detection class of a case can be one of: true positive, true negative, false positive,

and false negative. These correspond to the following possible monitoring outcomes: A true positive is an outcome where a relationship failure has actually occurred, and has been detected. A true negative is where no failure has occurred, and the system correctly reports none is being detected. A false positive is where no failure has occurred, but the system nevertheless incorrectly detects one, and a false negative is where a failure has occurred, but the system fails to detect it. Table 4.2 shows that in all permutations of Example 2 the teamwork monitoring techniques did not encounter the problematic false positive or false negative cases.

A closer look at these results hints at a key contribution of this dissertation in addressing the monitoring selectivity problem: Effective failure detection can take place despite the use of uncertain, limited, knowledge about monitored agents. In case 4 of Table 4.2, the monitoring agent was able to detect the failure *despite being wrong about the state of the agents involved.* The scout believes that the two attackers are executing the `H` (`ordered-halt`) plan, but they are actually executing `W`. `H` is selected when a command is received from headquarters to halt execution and hover in place. From the scout's perspective, a hovering attacker can therefore be inferred to be executing `H` or `W`. Thus two equally-ranked maximally-coherent hypotheses exist: the two attackers are either both executing `W` or both executing `H`. A random selection was made, and in this case resulted in the wrong hypothesis being selected. Nevertheless, a violation of the teamwork relationships was detected, as neither `H` or `W` agrees with the scout's `J`.

However, as the last column of case 4 shows (in Table 4.2), the diagnosis procedures are sensitive to the selection of the team-modeling hypothesis. The hypothesis used in this case does not correctly reflect the true state of the agents, and so despite the scout's success to detect a failure in this case, the diagnosis procedures fail to provide correct diagnosis (the diagnosis was successful in the two other failure cases). This phenomenon repeats in other empirical results we provide below: diagnosis failed whenever the hypothesis chosen was incorrect, although it was sufficient for detection. Ways to improve the accuracy of the hypotheses (for instance, for diagnosis) are discussed in Part II, and will not be addressed here further. The failure detection capabilities are a significant improvement in themselves, since the agents know with certainty that a failure has occurred, even if their diagnosis of it is incorrect.

Case #	Actual Executing Plans			Relationship Failure Occurred?	Physical Failure
	Attacker A1	Attacker A2	Scout A3		
1	W	W	W	-	-
2	F	W	W	+	A1 vision fails
3	W	F	W	+	A2 vision fails
4	F	F	W	+	A1, A2 vision fails
5	W	W	F	+	A3 vision fails
6	F	W	F	+	A1, A3 vision fails
7	W	F	F	+	A2, A3 vision fails
8	F	F	F	-	A1,2,3 vision fails

Table 4.3: All failure permutations of the undetected way-point scenario (Example 1).

Many social and physical failures are successfully captured using the team-coherence heuristic for monitoring selectivity. In fact, in our permutations of Example 2, no matter which one of the agents was the monitor, all failures to maintain relationship (i.e., all physical failures except for one in which the team remains in agreement) were detected reliably, although sometimes diagnosis failed. This result is especially surprising considering that only a single agent did the monitoring.

Tables 4.3 and 4.4 present further empirical results, on the basis of Example 1. Table 4.3 presents all failure permutations of Example 1 in the same format as Table 4.1. The normal order of execution of these plans is as follows: All agents jointly execute the `fly-flight-plan` (`F`) plan until they detect the way-point. They then switch to the `wait-at-point` (`W`) plan, in which the two attackers land while the scout continues to fly ahead to identify the enemy. The failures here are all on the part of the agents to detect the way-point, and thus to switch to the `W` plan.

Table 4.4 present the monitoring results for all permutations of Example 1. Here the attacker A1 is monitoring the team using again a maximally team-coherent hypothesis in detecting failures. The results show that A1 is successful in detecting all teamwork failures but two (cases 4-5, highlighted in bold face).

These two false outcomes are both false negatives. In both these cases, the monitoring attacker A1 picked an incorrect hypothesis for the scout, since the scout's actions lead to ambiguous interpretations. The scout is to fly forward (to scout the enemy) if it detected the way-point (plan `W`), but also if it did not (then it would

Case #	A1's Hypothesized Executing Plans			Relationship Failure Detected?	Detection Class
	Attacker A1	Attacker A2	Scout A3		
1	W	W	W	-	True Negative
2	F	W	F	+	True Positive
3	W	F	W	+	True Positive
4	**F**	**F**	**F**	-	**False Negative**
5	**W**	**W**	**W**	-	**False Negative**
6	F	W	F	+	True Positive
7	W	F	F	+	True Positive
8	F	F	F	-	True Negative

Table 4.4: Attacker's (A1) monitoring results in all permutations of Example 1.

be flying "in formation"–plan `F`). The use of the maximal team-coherence heuristic causes A1 to prefer a hypothesis in which the scout is in agreement with the attackers when in fact it is not. For example, in case 4, the two attackers have failed to detect the way-point and are executing `F`. Observing the scout, the monitoring attacker A1 is not sure whether the scout is executing `F` or `W`. However, believing that the scout is executing `F` results in a maximally-coherent team-modeling hypothesis (all the agents are in agreement), while believing that the scout is executing `W` results in a less coherent hypothesis. Thus A1 selects a wrong hypothesis, which in this case fails to detect the teamwork failure.

The maximal team-coherence heuristic can detect failures despite using incorrect hypotheses. Unfortunately, such hypotheses can also lead to false-negatives as we have seen in Table 4.4. However, none of our experiments resulted in a false-positive result, i.e., a result in which the system detected a failure but in reality none had occurred. Thus the heuristic provided sound results in these cases. We are able to formally prove this property holds in general when the maximal team-coherence heuristic is used.

First, we address a matter of notation. Let an agent A monitor an agent B, which is executing some plan P. We denote by $M(A, B/P)$ the set of agent-modeling hypotheses that A's agent-modeling component constructs based on B's observable behavior during the execution of P. In other words, $M(A, B/P)$ is A's set of all plans that match B's observable behavior. Note that when A monitors itself, it has direct access to its own state and so $M(A, A/P) = \{P\}$. Using the modeling

notation, we make the following definitions which ground our assumptions about the underlying knowledge used in monitoring:

Definition 3. Given a monitoring agent A, and a monitored agent B, we say that A's *agent-modeling* of agent B is *complete* if for any plan P that may be executed by B, $P \in M(A, B/P)$.

The set $M(A, B/P)$ will typically include other matching hypotheses besides the correct hypothesis P, but is guaranteed to include P. This definition abstracts several subtle issues. Complete modeling means we have a complete and correct plan-library, that our observations are correct, and that the observed actions have been executed intentionally by the monitored agent.

Following this definition of *individual* agent-modeling completeness, we can define group-wide team-modeling completeness:

Definition 4. Let A be an agent monitoring a team T of agents $B_1, \cdots, B_n$. We say that A's *team-modeling* of the team T is *complete* if A's agent-modeling of each of $B_1, \cdots, B_n$ is complete.

Definition 4 is critical to guarantee the capabilities we will explore analytically in this section and the next. It generally holds in our use of RESL in the ModSAF and RoboCup domains, and we make it explicit here in service of applications of the techniques in other domains.

Armed with these definitions, we now formalize the failure detection capabilities suggested by the empirical evidence in Theorem 1.

Theorem 1. *Let a monitoring agent A monitor a simple team T. If A's team-modeling of T is complete, and A uses a maximally team-coherent hypothesis for detection, then the teamwork failure detection results are sound.*

Proof. We will show that if no failure has occurred, none will be detected, and thus that any failure that is detected is in fact a failure. Let $a_1, \ldots, a_n$ be the agent members of T. Each agent a_i is executing some plan P_i $(1 \leq i \leq n)$. Thus collectively, the group is executing $(P_1, \ldots, P_n)$. If no failure has occurred, then all the agents are executing the same plan P_0, i.e., $\forall i, P_i = P_0$. Since A's team-modeling is complete, the correct hypothesis $(P_0, \ldots, P_0)$ is going to be in the set of team-modeling hypotheses H. Since it is a maximally team-coherent hypothesis, either it

Case #	A1's Hypothesized Executing Plans			Relationship Failure Detected?	Detection Class
	Attacker A1	Attacker A2	Scout A3		
1	**W**	**H**	**F**	+	**False Positive**
2	F	H	W	+	True Positive
3	W	F	F	+	True Positive
4	F	F	W	+	True Positive
5	W	H	F	+	True Positive
6	F	H	W	+	True Positive
7	W	F	F	+	True Positive
8	**F**	**F**	**W**	+	**False Positive**

Table 4.5: Attacker's (A1) monitoring results in all permutations of Example 1, using team-*incoherence*.

will be selected, or that a different hypothesis *of the same coherence level* will be selected. Any hypothesis with the same coherence level as the correct one implies no failure is detected. Thus the detection procedure is sound. □

Despite uncertainty in the knowledge used, sound failure-detection can be guaranteed using the maximal team-coherence heuristic. This is one answer to the monitoring selectivity problem. However, as we have seen in Table 4.4, some failures may pass undetected using this heuristic (i.e., it may result in false-negatives). Detection using maximal team-coherence may therefore unfortunately be *incomplete.* We may prefer our monitoring system to be *complete*—guaranteed to detect all teamwork failures.

We therefore experimented with the maximal team-*incoherence* heuristic, the inverse of the maximal team-coherence heuristic. This heuristic prefers hypotheses that suggest *more* failures, rather than less. Table 4.5 gives the monitoring attacker A1's view of the team, similar to Table 4.4, but using a maximally team-*incoherent* hypothesis. It shows that indeed using a maximally team-incoherent hypothesis will not lead to the false-negative detections in cases 4 and 5 (in contrast to these cases in Table 4.4).

Guided by these results, we formally show that the team-incoherence heuristic leads to a detection procedure that is *complete.*

Theorem 2. *Let a monitoring agent A monitor a simple team T. If the A's team-modeling of T is complete, and A uses a maximally team-*in*coherent hypothesis for detection, then the teamwork failure detection results are complete.*

Proof. Analogous to that of Theorem 1, the proof is provided in appendix A. □

However, these successes are offset by false positive outcomes in cases 1 and 8 of Table 4.5. In these cases, no failures have occurred, but the monitoring system falsely reported detected failures. In practice, this may lead to costly processing of many false alarms.

Ideally, the detection capabilities should be sound *and* complete. Unfortunately, we can show that no coherence-based disambiguation scheme exists that results in both sound and complete detection. We show in Theorem 3 that to provide sound and complete detection, a disambiguation method will have to be inconsistent: Given the same set of possible matching hypotheses, it will have to sometimes rank one hypothesis on top, and sometimes another.

Theorem 3. *Let H be a complete team-modeling hypotheses set, modeling a simple team. There does not exists a disambiguation scheme S that (1) uses coherence alone as the basis for disambiguation of H, and (2) is deterministic in its selection, and (3) results in sound and complete failure detection.*

Proof. Let S be a disambiguation scheme that leads to complete and sound detection and uses only knowledge of the coherence of the hypotheses in selecting a disambiguated hypothesis. Suppose for contradiction that it is deterministic, and thus consistent, in its selection of an hypothesis out of H, i.e., given H, a set of candidate hypotheses, it applies some deterministic procedure to choose one hypothesis based on its coherence. Since it does not use any other knowledge outside of the coherence of the candidate hypotheses, given the same set of candidates, it will always choose the same hypothesis. Let A_m be the monitoring agent using S. Let B be a monitored agent, whose actions are identical when executing team plans P_1, P_2. Thus, A_m cannot determine whether B is executing P_1 or P_2, $M(A_m, B/P_1) = M(A_m, B/P_2) = \{P_1, P_2\}$. If A_m and B are both executing P_1, A_m's hypotheses set is

$$H = \{(P_1, P_1), (P_1, P_2)\}$$

Since S leads to complete and sound detection, it will choose (P_1, P_1). Now, when A_m and B are executing P_1 and P_2, respectively, the matching hypothesis set is again H as defined above. But now S must select (P_1, P_2). Since the same set of candidate hypothesis H was used in each case, and no other information was supplied, S must be non-deterministic in its selection of a disambiguated hypothesis, contradicting the assumption. □

The empirical and analytical results show that our use of a single disambiguated hypothesis leads to improved, but imperfect, failure-detection results, compared to the monitoring conditions and communications previously used. The empirical results in Tables 4.2, 4.4, and 4.5 establish the benefits of the teamwork monitoring technique: Most physical failures were detected. However, the analytical results (Theorems 1, 2, 3) show that the results are less than perfect. The algorithms are either sound or complete, but not both. For complete monitoring, we would require additional procedures that can differentiate the true positives from the false ones, e.g., by focused communication. These procedures are often very expensive.

We can reduce the need for costly verification by letting go of our insistence on a single hypothesis, focusing instead on maintaining two hypotheses: a maximally-coherent hypothesis and a maximally-incoherent hypothesis. Table 4.6 shows a portion of the full set of team-hypotheses available when the attacker A1 is monitoring the team. The total number of hypotheses presented in the table is 24, with as many as 4 co-existing in a single case, and thus maintaining a full set of hypotheses would be expensive. However, the two inverse heuristics—team-coherence and incoherence—represent two extremes of the space of these hypotheses. If they agree that a failure exists, then a failure actually occurred, since the team-coherent hypothesis guarantees soundness (Theorem 1). If they agree that no failure exists, then no failure took place, since the team-incoherent hypothesis guarantees completeness (Theorem 2). If they disagree (i.e., the team-coherent hypothesis does not imply a failure, but the team-incoherent hypothesis does), the monitoring system cannot be sure either way, and must revert back to verification.

This revised detection algorithm offers significant computational savings compared to the single team-incoherent hypothesis approach. It is complete and unsound, but significantly reduces the need for verification, since at least when the team-coherent hypothesis implies failures, verification is not necessary. It requires

Case	A1's Hypothesized Executing Plans			Relationship Failure	Detection
#	Attacker A1	Attacker A2	Scout A3	Detected?	Class
1	W	H	F	+	False Positive
	W	H	W	+	False Positive
	W	W	F	+	False Positive
	W	W	W	-	True Negative
2	F	H	F	+	True Positive
	F	H	W	+	True Positive
	F	W	F	+	True Positive
	F	W	W	+	True Positive
3	W	F	F	+	True Positive
	W	F	W	+	True Positive
4	F	F	W	+	True Positive
	F	F	F	-	False Negative
5	W	H	F	+	True Positive
	W	H	W	+	True Positive
	W	W	F	+	True Positive
	W	W	W	-	False Negative
6	F	H	W	+	True Positive
	F	H	F	+	True Positive
	F	W	W	+	True Positive
	F	W	F	+	True Positive
7	W	F	F	+	True Positive
	W	F	W	+	True Positive
8	F	F	W	+	False Positive
	F	F	F	-	True Negative

Table 4.6: A portion of the attacker's (A1) monitoring hypotheses and implied results when no ranking is used to select a single hypothesis for each case.

representing only two hypotheses, and is thus still computationally cheaper than maintaining an exponential number of hypotheses.

For example, using a maximally team-incoherent hypothesis on permutations of Example 1 results in a need to verify in all eight cases as we have seen (4.5). However, when we combine such an hypothesis with a maximally team-coherent hypothesis (e.g., as in Table 4.4), we only need to verify four (50%) of the cases. In cases 2, 3, 6, 7 there is agreement between the two hypotheses that a failure has occurred, and thus no verification is required.

A monitoring agent can therefore address the monitoring selectivity problem by balancing its resource usage against the guaranteed performance of the monitoring algorithm used. Either of the simpler single-hypothesis algorithms would utilize only one hypothesis in each case, with detection capabilities that are guaranteed to be sound or complete, but not both. In the more complex algorithm, two hypotheses would be reasoned about in each case, and the algorithm would be complete and require verification in fewer cases compared to the simple-hypothesis complete algorithm.

Chapter 5

Monitoring Selectivity in Distributed Teamwork Monitoring

This chapter focuses on monitoring selectivity when exploiting a key opportunity for execution monitoring in multi-agent environments—it is not only the monitored agents that are distributed, but the monitoring agents can be distributed as well. We begin with the simple scheme of selecting a single maximally team-coherent hypothesis. Since centralized teamwork monitoring was successful in addressing all permutations of Example 2, we focus here on the permutations of Example 1 (Table 4.3), in which centralized teamwork monitoring by the attacker resulted in false-negative detections (cases 4-5 in Table 4.4).

In a distributed teamwork monitoring scheme, not only will a single attacker monitor its teammates, but the scout (and the other attacker) will also engage in monitoring. Table 5.1 presents the monitoring results of the same failure permutations, with the scout as the monitoring agent. We find that the scout successfully detects the two failure cases that the attacker failed to detect, compensating for the attackers' monitoring mistakes. Furthermore, since the scout used the the maximal-coherence heuristic, detection is sound and no verification is required. The reason for the scout's success is that the attackers' actions in this case, although ambiguous, do not support any hypothesis that can be matched to the scout's plan. In other words, regardless of what plan the attackers are executing in these two cases, it is different than the plan executed by the scout.

Thus if all agents engaged in monitoring in permutations of Example 1, detection would be sound and complete. In all actual failure cases (and only in those) there

Case #	A3's Hypothesized Executing Plans			Relationship Failure Detected?	Detection Class
	Attacker A1	Attacker A2	Scout A3		
1	W	W	W	-	True Negative
2	F	W	F	+	True Positive
3	W	F	W	+	True Positive
4	F	F	W	+	True Positive
5	H	H	F	+	True Positive
6	F	H	F	+	True Positive
7	H	F	F	+	True Positive
8	F	F	F	-	True Negative

Table 5.1: Scout's (A3) monitoring results in all permutations of Example 1, using team-coherence.

would at least one team-member who detects the failure. We attempt to formally define the general conditions under which this phenomenon holds.

Definition 5. We say that two team-plans P_1, P_2, have *observably-different roles* R_1, R_2 if given an agent B who fulfills the roles R_1, R_2 in the two plans, respectively., any monitoring agent A (different than B) will have $M(A, B/P_1) \cap M(A, B/P_2) = \emptyset$. We then say that B has observably-different roles in P_1 and P_2, and call B a *key agent.*

Intuitively, B is a key agent that has observably different roles in the two plans if a monitoring agent can differentiate between B's behavior in executing P_1 and in executing P_2. For instance, both attackers have observably different roles in `F` (in which they fly) and `W` (in which they land). However, they do not have observably different roles in `W` and `H`, both of which require them to land. The scout has observably different roles in `W` (flying) and `H` (landing).

The key agent is the basis for the conditions under which a self-monitoring team will detect a failure with each agent using only team-coherence. We first prove a lemma on the conditions in which a single given agent will detect a failure. We then use this lemma to prove the conditions under which at least one agent in a given team will detect a failure.

Lemma 1. *Suppose a simple team T is self-monitoring (all members of the team monitor each other) using the maximally team-coherent heuristic (and under the assumption that for each agent, team-modeling is complete). Let A_1, A_2 be monitoring*

agents who are members of T and are executing P_1, P_2, respectively. A_1 would detect a failure in maintaining teamwork relationships with an agent A_2 if A_2 is a key agent in P_1, P_2.

Proof. See appendix A. □

A_1 knows that it is executing P_1. If A_2 is executing P_2, and is a key agent in P_1 and P_2, then A_1 is guaranteed to notice that a difference exists between itself and A_2, since A_2 is acting observably different than it would if it had been executing P_1. Note, however, that A_2 may or may not detect this difference, since from A_2's perspective, A_1's behavior may or may not be explained by P_2. A_2 will detect a difference only if A_1's roles in P_1 and P_2 are also observably-different. However, since A_1 has detected the failure, it can now alert its teammates, diagnose the failure, or choose corrective action.

If we want to guarantee that a teamwork failure will always be detected by at least one agent, we must make sure that in each possible combination of plans, there has to be at least one key agent whose roles are observably different. The lemma shows that other agents monitoring this agent will notice a failure if one occurs. To this aim, we define an observably-partitioned set of plans employed by a team.

Definition 6. A set P of team-plans is said to be *observably-partitioned* if for any two plans $P_i, P_j \in P$ there exists a key agent A_{ij}. The set of these A_{ij} agents is called the *key agents set of* P.

For instance, the set of team-plans our helicopter pilots team has been using in the examples (`Fly-Flight-Plan` (`F`), `Wait-at-Point` (`W`), `Ordered-Halt` (`H`), and `Join-Scout` (`J`)) is observably-partitioned. The attackers land in `W` and `H`, but fly in `F` and `J`. The scout lands in `J` and `H`, but flies in `W` and `F`. Table 5.2 shows which agents have observably different roles in any two plans in the set. For instance, by finding the cell at the intersection of the `H` row and the `W` column, we find that the scout has observably different roles in these two plans. Indeed, the scout lands when a command is received to halt execution (`H`), but flies out to scout the enemy's position when executing `W`. Here, since all agents have observably-different roles in at least two plans, the key agents set of { `W`, `F`, `H`, `J` } includes all members of the team—attackers and scout.

	Fly-Flight-Plan (F)	Wait-at-Point (W)	Ordered-Halt (H)	Join-Scout (J)
F	-	Attackers	Attackers	Scout
W	Attackers	-	Scout	Scout and Attackers
H	Attackers	Scout	-	Attackers
J	Scout	Scout and Attackers	Attackers	-

Table 5.2: Observable partitioning of the helicopter pilot team in ModSAF.

Theorem 4. *If a simple team (1) employs an observably-partitioned set of team-plans O, and (2) all team-members monitor* members of the key agents set of O, *(3) using complete team-modeling and (4) maximally team-coherent hypotheses, then the teamwork failure detection results are sound and complete.*

Proof. From theorem 1 we know that detection would be sound. To show that it is complete, we will prove at least one agent will detect a difference between itself and others whenever team-members are not all executing the same plan (i.e., a failure is occurring). Suppose the team is currently divided on the team-plans that must be executed, i.e., there are agents a_i, a_j in the team that are executing team plans P_i, P_j, respectively, such that $P_i \neq P_j$. Thus a failure has occurred. Let K be the key agents set of O. Since the team is observably-partitioned, for P_i, P_j there exists at least one key agent $a_1 \in K$. There are three cases:
case (i). a_1 is executing P_i. In this case any agent executing P_j would detect a difference with a_1 and would therefore detect the failure (lemma 1).
case (ii). a_1 is executing P_j. In this case any agent executing P_i would detect a difference with a_1 and would therefore detect the failure (lemma 1).
case (iii). a_1 is executing some other plan Q. Its roles must be observably different in Q and P_i, or in Q and P_j (or in both), and thus any agent executing P_j and/or P_i would detect the failure. The case that a_1's roles are not observably different in Q and P_i *and* in Q and P_j is impossible, since then for a monitoring agent A_m

$$M(A_m, a_1/P_i) \cap M(A_m, a_1/Pj) \supseteq \{Q\} \neq \emptyset$$

Contradicting a_1 being a key agent for P_i, P_j.
Since in all three cases, at least one agent would detect a failure where one occurred. Therefore, failure detection is complete. Since it is also sound as we have seen, detection is sound and complete. □

The theorem shows that distributed teamwork monitoring can result in *sound and complete* failure-detection, while using a simple algorithm. Each team-member monitors *only the key agents*[1], using a maximally team-coherent hypothesis. If it detects a failure, then certainly one has occurred. If no agent detects a failure, then indeed no failure has occurred.

This simple distributed algorithm, with its attention-focusing features and guaranteed soundness and completeness contrasts with the more complex centralized algorithm which we discussed in the previous section (Chapter 4). The algorithm's effectiveness relies on the condition of an observably-partitioned set of plans, and the distribution of the monitoring. This distribution allows agents to exploit their local state here, which is not available to the centralized algorithm. Indeed, even when key agents are available, centralized teamwork monitoring is still not complete and sound. A corollary of Theorems 3 and 4 is that if key agents are not available in the distributed case, failure detection is either sound or complete, but not both.

Fortunately, observable-partitioning is not a difficult property to design: Teams are very often composed such that not all agents have the same role in the same plan, and in general, roles do have observable differences between them. For instance, our helicopter pilot team in the ModSAF domain typically executes a set of plans with this property, as Table 5.2 demonstrates.

If the team, however, is not observably-partitioned, there may be a case where two agents are each executing a different plan, but no agent will be able to detect it using the team-coherence heuristic. The minimal case where this occurs is when two agents, A_1 and A_2 are executing plans P_1 and P_2, respectively, and P_1 and P_2 are not observably different, such that

$$M(A_2, A_1/P_1) \cap M(A_1, A_2/P2) = \{P_1, P_2\}$$

This will result in A_1 and A_2 each believing that the other is in agreement with them. A check for such a situation can be made a part of the plan design process, marking *risky points* in the execution in which detection is either sound or complete (Theorem

[1]If the monitoring team-member does not know who the key agents are, but knows they exist, it can monitor all other team-members. This increases monitoring, but sound and complete failure detection is still guaranteed.

3), and verification (e.g., by communications) can be prescribed pro-actively. Or, the check could be inserted into the protocol for run-time analysis—the agent would simulate the other's hypotheses matching their own actions, and detect risky points dynamically.

Chapter 6

Using Socially-Attentive Monitoring in an Off-Line Configuration

To further demonstrate the generality of our socially-attentive monitoring framework (Chapter 3), this chapter examines re-use of teamwork monitoring in domains in which diagnosis and recovery from every failure are infeasible during execution. Examples of such domains include team sports, military human team training (Volpe, Cannon-Bowers, & Salas, 1996), and other multi-agent domains. The dynamic nature of the domain, hard real-time deadlines, and complexity of the agents involved (e.g., human team-members) make diagnosis and recovery difficult. Even if a failure can be diagnosed, it is often too late for effective recovery. In such environments, the monitoring agent is often concerned with trends of performance. This information is important for long-term design evaluation and analysis, and need not necessarily be calculated on-line. The results of the analysis are meant as feedback to the agents' designer (coach or supervisor, for humans).

To this end, we developed an off-line socially-attentive monitoring system called TEAMORE (TEAmwork MOnitoring REview). TEAMORE currently uses execution traces of the monitored agents to perform the monitoring, rather than using plan recognition. Thus it does not need to worry about the uncertainty in plan recognition, nor about real-time performance. Instead, it knows with certainty each agent's plans during execution. TEAMORE accumulates several quantitative measures related to teamwork, including the Average-Time-to-Agreement measure (ATA, for short), and a measure of the level of agreement in a team. These build on the failure

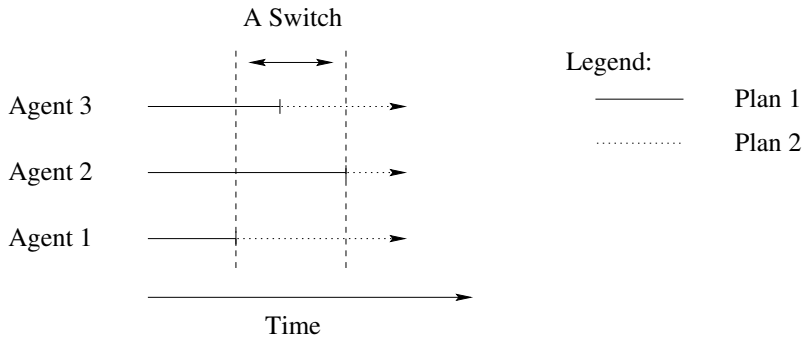


Figure 6.1: An illustration of a switch. The three agents switch from plan 1 to plan 2.

detection algorithm, but aggregate failures in quantitatively. We focus here on the ATA measure.

TEAMORE defines a *switch* as the time interval beginning at the point where any team-member (at least one) selects a new team plan for execution by the team, and ending at the point where the team is again in agreement on the team-plan being executed. In perfect teamwork, all team-members select a new team-plan jointly, and so always remain in agreement. In a more realistic scenario, some agents will take longer to switch, and so initially a teamwork failure will occur. The first team-member to select a new plan will be in disagreement with some of its teammates, until either it rejoins them in executing the original plan, or they join it in selecting the new plan. Such a switch begins with a detected failure and ends when no more failures are detected.

Figure 6.1 shows an illustration of a switch. The three agents begin in an initial state of agreement on joint execution of Plan 1 (filled line). Agent 1 is the first agent to switch to Plan 2 (dotted line), and is followed by Agent 3, and finally Agent 2. The switch is the interval which begins at the instance Agents 1 selected Plan 2, to the time all three agents regained their agreement (but this time on Plan 2).

TEAMORE keeps track of the lengths of time in which failures are detected until they are resolved. The ATA measure is the average switch length (in time "ticks") per a complete team run (e.g., a mission in ModSAF, a game in RoboCup). A perfect team would have all switches of length zero, and therefore an ATA of 0. The worst team would be one that from the very beginning of their task execution to the very end of it, would not agree on the team plan being executed. For instance, each

ISIS sub-team	Mean ATA No comm.	Mean ATA Comm.	ATA Reduction	t-test prob. null-hypothesis
'97 Goalies	32.80	5.79	27.01	7.13e-13
'97 Defenders	57.5	6.81	50.69	.45e-10
'98 Goalies	13.28	3.65	9.63	9.26e-16
'98 Defenders	12.99	3.98	9.01	7.13e-5

Table 6.1: Average-Time-to-Agreement (ATA) for games against Andhill'97.

RoboCup game lasts for 6000 "ticks". The worst possible team would have only one switch during the game, of length 6000. Thus the ATA scale in RoboCup goes from 0 (perfect) to 6000 (worst).

We used the ATA measure to analyze a series of games of our two RoboCup simulation-league teams, ISIS'97 and ISIS'98 (Marsella et al., 1999) against a fixed opponent, Andhill'97 (Ando, 1998). In these games, we varied the use of communications by our teams to evaluate design decisions on the use of communications. In approximately half of the games, players were allowed to use communications in service of teamwork. In the other half, all communications between agents were disabled. ISIS'97 played approximately 15 games in each settings, and ISIS'98 played 30 games in each communication settings.

Table 6.1 shows the mean ATA values over these games, for two sub-teams (each having three members) of ISIS'97 and ISIS'98 (ATA values are calculated separately for each sub-team). The first column shows which sub-team the results refer to in each row. The second columns shows the mean ATA for each sub-team, when no communications were used. The third column shows the mean ATA when communications were used. The next column shows the size of the ATA reduction—the drop in the mean ATA values when communications are introduced. The last column shows the probability of the null hypothesis in a two-tailed t-test of the difference in the ATA means. This is the probability that the difference is due to chance, thus smaller numbers indicate greater significance.

Clearly, a very significant difference emerges between the communicating and non-communicating versions of each sub-team. The ATA values indicate that sharing information by way of communications significantly decreases the time it takes team-members to come to agreement on a selected plan. This result agrees with our intuitions about the role of communications, and in that sense, may not be surprising.

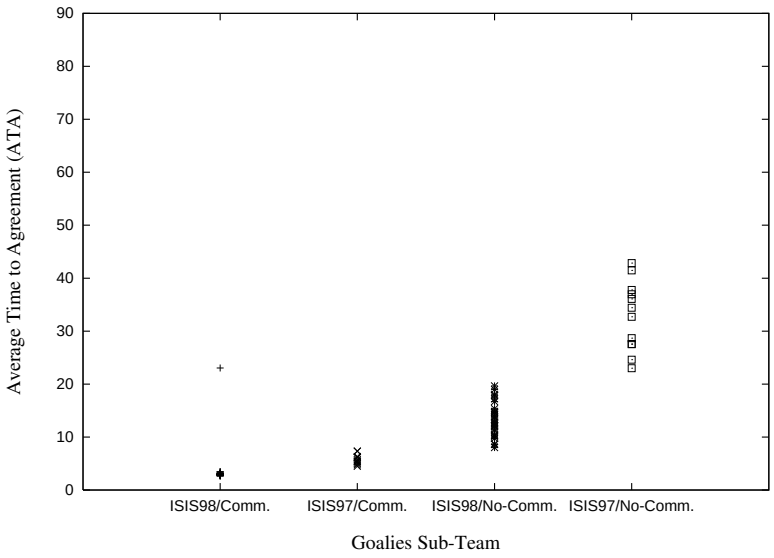
	ISIS'97	ISIS'98
Communication Used	-3.38	-1.53
Communication Not Used	-4.36	-2.13
t-test p/null hypothesis	p=0.032	p=0.13

Table 6.2: ISIS'97 and ISIS'98 mean score difference against Andhill'97, with changing communications settings

However, the ATA reduction magnitudes indicate that ISIS'98 may be much less sensitive to loss of communications than ISIS'97. The differences in ATA values for ISIS'97 are approximately triple, nearly four times, as great as for ISIS'98. Our explanation for this phenomenon is that ISIS'98 is composed of players with improved capabilities for monitoring the environment (such that they have better knowledge of the environment). ISIS'98 is therefore not as dependent on communications as are teams, such as ISIS'97, composed of players with lesser environment monitoring capabilities. ISIS'98 players are better able to select the correct plan without relying on their teammates. Thus, they would be able to maintain the same level of performance when communications are not used. In contrast, ISIS'97 players rely on passing information to and from each other (monitoring each other) through communications, and so took much longer to establish agreement when communications were not available.

We can validate the hypothesis suggested by ATA measurements by looking at the overall team-performance in the games, measured by the score difference at the end of the game. Table 6.2 shows the mean score difference from the same series of games against Andhill'97. The first column lists the communications settings (with or without). The second and third columns show the *mean* score-difference in the games for ISIS'97 and ISIS'98. The bottom row summarizes the results of t-tests run on each set of games, to determine the significance level of the difference between the mean score-differences. The score-difference results corroborate the ATA results. While the difference in mean score-difference is indeed statistically significant in ISIS'97 games, it is not significant in ISIS'98 games. This supports our explanation that the more situationally aware ISIS'98 is indeed better able to handle loss of communications than ISIS'97.

The general lesson emerging from these experiments is that a trade-off exists in addressing the monitoring selectivity problem. The knowledge that is maintained



(a) ATA Values for Goalies subteam

Figure 6.2: ATA values for the Goalies sub-teams in games against Andhill'97.

about teammates (here, via communications) can be traded, to an extent, with knowledge maintained about the environment. A designer therefore has a range of alternative capabilities that it can choose for its agents. Different domains may better facilitate implicit coordination by monitoring the environment, while others require agents to rely on communications or explicit knowledge of team-members to handle the coordination.

The ATA results support additional conclusions, especially when combined with a general performance measure such as the score-difference. To illustrate, consider the plots of the actual data from these games. Figure 6.2 plots all the ATA values for all four variants, for the Goalies sub-team. The graph plots approximately 60 data-points. We see in Figure 6.2 that when communications are used, ISIS'97's ATA values are still generally better than ISIS'98's ATA without communications. Thus, despite its importance, individual situational awareness is not able to fully compensate for lack of communications.

TEAMORE demonstrates the reuse of the teamwork monitoring techniques developed in earlier sections in an off-line configuration. The designer of ISIS'97 should set its agents to use communications, since those will have significant improvement

on the score-difference. In contrast, with or without communications, ISIS'98 players are able to maintain their collaboration. Thus if communications takes precious resources, it can be relatively safely eliminated from the ISIS'98 agents' design, and the development efforts can be directed at some other components of the agents.

Chapter 7

Beyond Teamwork Failures

We have presented a general socially-attentive monitoring framework to detect failures in maintaining agreement on joint team plans. However, effective operation in teams often relies on additional relationships, which we briefly address in this chapter.

7.1 A Richer Agreement Model: Agreeing to Disagree

The teamwork model requires joint execution of team plans. In service of such agreed-upon joint plans, agents may sometimes agree to execute different sub-plans individually, or split into sub-teams to execute different sub-team plans. Two examples may serve to illustrate.

Example 3. In the ModSAF domain, helicopters engage the enemy by repeatedly performing the following three steps: hiding behind a hill or trees (*masking*), then popping up (*unmasking*), then shooting missiles at the enemy, and back to hiding. In some variations of this plan, they are required to make sure that no two helicopters are shooting at the same time. Of course, due to limits of communications, helicopters do fail and unmask at the same time.

Example 4. In the RoboCup domain, our 11 players in both ISIS'97 and ISIS'98 (Marsella et al., 1999) are divided into four sub-teams: mid-fielders, attackers, defenders, and goalies (the goalie and two close defenders). This division into sub-teams is modeled by the agents selecting one of four team plans in service of the

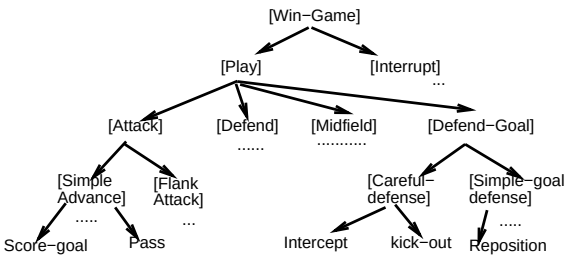


Figure 7.1: A Portion of the plan hierarchy used by ISIS RoboCup agents.

`play` team plan (see Figure 7.1). Mid-fielders must select the `midfield` plan, goalies must select the `defend-goal` plan, etc. Again, ideally an attacker would never select any other plan but `attack`, a defender would select no other plan but `defend`, etc. However, due to communication failures, players may sometimes accidently abandon their intended sub-team, and execute a team-plan of another sub-team.

In both of these examples, certain differences between agents are agreed upon and are a sign of correct execution, not of failure. Indeed, it is the lack of difference in selected plans that would indicate failure in these cases. We use the term *mutual-exclusion coordination* to refer to these relationships. In Example 3, ideally no two pilots are executing the `shooting` plan at the same time. In Example 4, no two members of different sub-teams (e.g., an attacker and a defender) are executing the same plan in service of `play` (e.g., `defend`). As the examples demonstrate, there is a clear need for monitoring mutual-exclusion coordination.

Our results of previous sections are re-used in service of socially-attentive monitoring of mutual-exclusion relationships. They require a transformation both in implementation and theory. The hierarchies are compared in the usual manner, except that failures are signified by equalities, rather than differences. For instance, if an attacker is staying in the team's own half of the field, its teammates may come to suspect that it mistakenly "defected" the attackers' sub-team and believes itself to be a defender.

The analytical results are inverted as well. The maximal team-coherence heuristic will now lead to completeness, since it prefers hypotheses that contain equalities among agents, which are failures in mutual-exclusion coordination. The maximal team-incoherence heuristic will now lead to sound detection, as it prefers hypotheses that imply no equalities have occurred. These properties can be proven formally.

Theorem 5. *Let a monitoring agent A monitor mutual-exclusion relationships in a group of agents G. If A's modeling of G is complete, and A uses a maximally team-incoherent hypothesis for detection, then the failure detection results are sound.*

Proof. Provided in appendix A. □

Theorem 6. *Let a monitoring agent A monitor mutual-exclusion relationships in a group of agents G. If A's modeling of G is complete, and A uses a maximally team-coherent hypothesis for detection, then the failure detection results are complete.*

Proof. Provided in appendix A. □

Thus in mutual-exclusion relationships, as in teamwork relationships, guaranteed failure-detection results may still be provided despite the use of limited, uncertain knowledge about monitored agents. The centralized teamwork monitoring algorithms can now be easily transformed for monitoring mutual-exclusion relationships. Unfortunately, the results in the distributed case (Theorem 4) cannot be so easily transformed, since they rely on the property of observable-partitioning, which is associated with differences, not with equalities. We leave this issue for future work.

7.2 Monitoring Using Role-Similarity Relationships

This section applies socially-attentive monitoring to role-similarity relationships, for monitoring individual performance within teams. In particular, in service of team-plans agents may select individual sub-plans, which do not necessitate agreement by team-members, but are constrained by the agents' roles. For instance, in service of executing the team-plan `fly-flight-plan` (Figure 3.2) pilots individually select their own individual plans which set the velocity and heading within the constraints of the formation and flight method specified in the mission.

Role similarity relationships specify the ways in which given individual plans are similar, and to what extent. Two agents of the same role who are executing dissimilar plans can be considered to be in violation of the role-similarity relationships. This enables a socially-attentive monitoring system to detect failure in role-execution. To monitor individual plans the agent is executing, it compares its selection with that

of other agents of *the same role*, similarly to the method we used for teamwork. If the plans are considered similar by the role-similarity relationship model, no failure is detected. Otherwise, a failure may have occurred, and the diagnosis component is called to verify it and provide an explanation.

Let us illustrate with a failure from the ModSAF domain which our system was able to detect using the role-similarity relationship:

A team of three helicopters was to take off from the base and head out on a mission. However, one of the pilot agents failed to correctly process the mission statement. It therefore kept its helicopter hovering above the base, while its teammates left to execute the mission by themselves.

This failure was detected using role-similarity relationship monitoring. The agreed-upon team-plan was selected by all the agents, and so no problem with teamwork relationship was detected. This team-plan involved each agent then selecting individual methods of flight, which determine altitude and velocity. Here the agents differed. The failing helicopter remained hovering, while its teammates moved forward. Using a role similarity relationship, the failing helicopter compared its own selected plan to that of its teammate (who shared its role of a subordinate in the formation), and realized that their plans were dissimilar enough to announce a possible failure.

Unfortunately, the actual similarity metrics seem to be domain- and task-specific, and thus are not as easy to re-use across domains. Furthermore, detected failures are not necessarily real failures, nor do all detected failures have the same weight. We are currently investigating ways to address these challenging issues.

Part II

Monitoring Distributed Teams

Chapter 8

Motivation and Background: Monitoring Distributed Teams

In this part we examine a different monitoring task, that of on-line identification of the state of a distributed team. The monitoring selectivity problem is more pronounced in this task, as the distribution of the team may make most (local) actions of an agent unobservable to the monitoring system, leading to much greater uncertainty about monitored agents' state. Furthermore, the goal of the task is accurate state identification (to the degree possible), rather than failure/no-failure decision.

One key approach to tackling this challenge is to use inter-agent communications as the basis for the monitoring; i.e., by eavesdropping on the organization's internal communication (Ndumu et al., 1999), since communications are easier to eavesdrop on from a single location (when broadcast communications are used), or from a handful of places (when point-to-point messages are routed through the network). Communication-based monitoring is also important because: (a) it can be completely non-intrusive, not requiring agents to modify their existing behaviors; and (b) the growth of agent development and integration architectures (Ndumu et al., 1999; Pynadath et al., 1999) leads to increased standardization of agent communications, providing opportunities for such monitoring.

Unfortunately, eavesdropping on communications does not eliminate the uncertainty in monitoring. team-members cannot and do not in practice continuously communicate about all their on-going plans and actions (Jennings, 1995; Pechoucek et al., 2000). Furthermore, communications sometimes occur in small sub-teams,

and yet the monitoring system must infer the state of the entire team and other sub-teams. The limited communications serving as the basis for monitoring result in significant uncertainty in inferring the teams' on-going plans. This uncertainty is further exacerbated when we consider that agents may unexpectedly fail, as the monitoring system must then predict how other agents will respond to the failures.

Concretely, the motivation for our exploration of communications-based monitoring comes from our work in building a system for rehearsing the evacuation of civilians from a threatened location. The 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 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 (typically 11): Quickset (Multimodal command input agents, OGI), Route planner (Path planner for aircraft, CMU), Ariadne (Database engine for dynamic threats, USC/ISI), and Helicopter pilots (Pilot agents for simulated helicopters , USC/ISI). Generally, the actions of agents are not observable to each other. For instance, the dialog management actions of the Quickset agents are only observable by the user who inputs the commands; the route-planning actions are only reported on the screen of the computer running the route-planner, etc.

The application was built using the TEAMCORE multi-agent integration architecture (Pynadath et al., 1999). Each single domain agent is assigned a TEAMCORE proxy. The proxies jointly execute a *team-oriented program*, consisting of a set of hierarchical team plans, with assigned roles for teams and sub-teams (approximately 60 team plans are used in this domain). As an example, Figure 8.1-a shows a part of the team/sub-team hierarchy used in the evacuation-domain (described below). Here, for instance, TRANSPORT is a sub-team of Task-Force. Figure 8.1-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

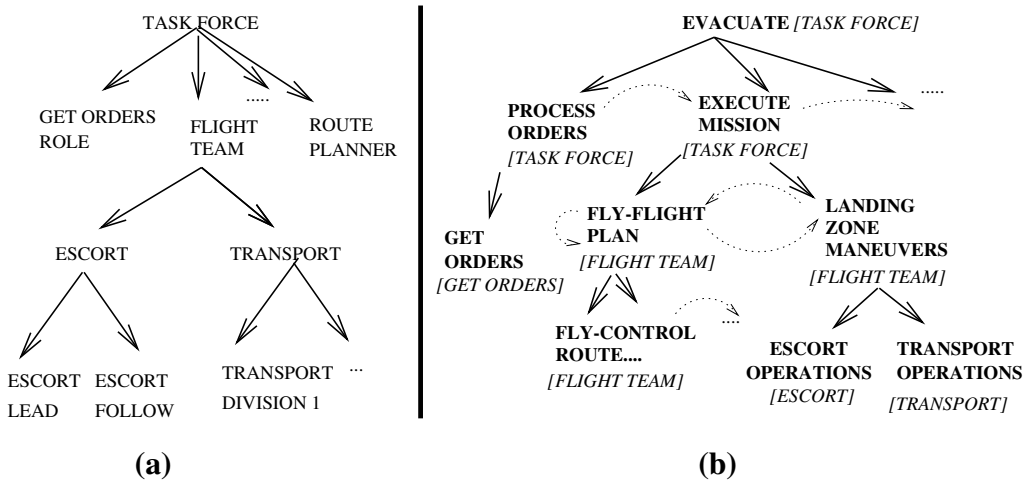


Figure 8.1: Portions of the team-hierarchy (a) and plan hierarchy (b) used in our domain. Dotted line show temporal transitions.

an in-built domain-independent teamwork model, called STEAM (Tambe, 1997). STEAM allows the agents to automatically coordinate using selective communications, asking each other to jointly terminate or initiate team-plans.

Humans and agents must monitor the resulting team based on these communications, answering queries about the present and future likely states of the entire team, its sub-teams and individuals—to monitor progress, compute likelihoods of failure, etc. 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 sub-team will fail in fulfilling their role — to take remedial actions if this likelihood is high.

This is a challenging task because: (i) only some agents communicate, and only about some plans; and (ii) agents may occasionally fail, but these failures cannot in general be observed. Simple visualization tools were available to provide some clues—such as the last message received from an agent, or the physical location of the helicopter agents—but these prove insufficient in being able to accurately answer the queries.

As a basis for monitoring, we built on an efficient single-agent probabilistic plan recognition representation and algorithm[1]. The key idea in this representation is that

[1]This algorithm was co-developed with David V. Pynadath, and is described fully in Appendix C.

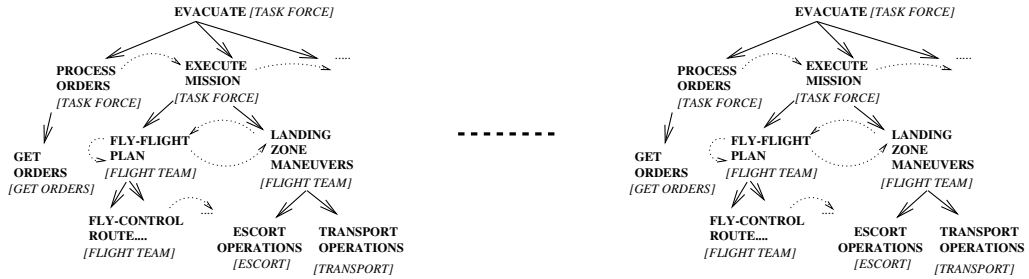


Figure 8.2: Array of single-agent recognizers—one for each agent.

the team-oriented plan is represented as a directed graph, whose vertices are plans, and whose edges signify temporal and hierarchical decomposition transitions between plans. The algorithm uses temporal knowledge about the average duration of plans, to attempt to predict an agent's current selected plan based on the last plan(s) it has been known to execute (i.e., the plan(s) implied by the last communication message received from the agent), and the time that has passed since that last plan was observed. For instance, given that the message "`Process-Orders` was initiated" was received 5 seconds ago, and that on average it takes only 3 seconds to execute, the algorithm will return the probability that the agent has terminated `Process-Orders` and initiated the next plan in line (as well as returning the probability that it had stayed).

The initial multi-agent monitoring solution uses an array of such single-agent recognizers, one for each agent (illustrated in Figure 8.2). Messages are overheard in parallel (when agents communicate simultaneously). The idea is that each message is used as evidence for the agent that sent it. The state of agents for which no message is received is updated solely based on the temporal model. 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 proved insufficient. First, the cost of maintaining a plan hierarchy for each agent is prohibitive as the number of agents in the team increases. The array of recognizers includes 726 plans to reason about the 11 agents in the evacuation rehearsal scenario, with the number of plans increasing by 66 plans with each additional agent. Second, the accuracy in identifying the correct state of the team as the most likely hypothesis *was under 4% in all experiments.*

These poor accuracy results are mostly due to the high uncertainty caused by the limited observations (intercepted communications) and uncertainty in the temporal model used as the basis for inference. The limited observations result in multiple hypotheses as to the state of each agent. As these individual-state hypotheses are combined, they lead to a potentially exponential number of team-state hypotheses (as previously discussed in Section 3.2). Indeed, with 11 agents, each with just two possible individual hypotheses, there are more than 2000 possible hypotheses as to the state of the team. The temporal model used for selecting the most likely one based on average plan durations cannot be made accurate enough to correctly choose the correct hypothesis in all cases.

The monitoring selectivity problem is thus raised here again. Here, the limited observations impose a-priori high selectivity in observations, resulting in much uncertainty about team-members' states. Given this uncertainty, it is difficult to correctly identify the correct team-state. Furthermore, reasoning about an exponentially-increasing number of hypotheses makes the task computationally expensive, requiring further selectivity in inference to make monitoring practical.

The monitoring task here is different than the failure-detection monitoring task discussed in Part I: (a) Unlike monitoring for failure-detection, we are interested in the full correct state of the team; and (b) the levels of uncertainty are much higher, since most agents' actions are unobservable.

The questions we address in this part stem from this difference in the monitoring task. The first part of this dissertation has focused on showing that certain monitoring goals, such as relationship failure detection, can be guaranteed despite selectivity (and has shown that in the distributed case higher levels of selectivity—not monitoring all agents—can be tolerated). This part presents socially-attentive methods to mitigate the uncertainty resulting from selectivity (Chapter 9), and the computational complexity involved in reasoning about this uncertainty (Chapter 10). The methods were implemented in a system called `Eavesdropper`, and evaluated in the evacuation rehearsal domain.

Previous work on monitoring multi-agent (e.g., Intille & Bobick, 1999; Devaney & Ram, 1998) have focused largely on mechanisms utilizing the known multi-agent plan-libraries, without utilizing deeper knowledge about the social relationships which the agents maintained. For instance, such techniques do not exploit

knowledge about future steps that the agents may take in service of maintaining their relationships. Chapter 11 provides detailed treatment of related work.

Chapter 9

Exploiting Teamwork Knowledge for Accurate Monitoring

This chapter examines closely the uncertainty issue in the monitoring selectivity problem. We identify three areas of uncertainty in this monitoring task, all resulting from the limited observations available to the monitoring agent (here, the observations are of communicated messages): First, there is uncertainty about the time that plans take to execute, which leads to uncertainty about which plans have already been executed (or have begun execution) by the time of the query. Second, after each plan, the agent may be free to choose its next plan step from a number of alternatives, which leads to uncertainty about the agent's chosen path of execution among alternative paths (each of them possibly involving uncertainty of the first type). Third, there is uncertainty about the relative states of execution of different agents, which may or may not have failed to maintain their ideal states relationships.

We report on two socially-attentive monitoring techniques for tackling the monitoring selectivity problem, by directly attacking these areas of uncertainty. The first technique revisits the minimal-number-of-failures heuristic used in Part I, using knowledge of the relationships in the monitored distributed team to prefer hypotheses in which the relationships in the organization are not violated. The heuristic is useful in alleviating uncertainty in all three areas of uncertainty. The second technique uses knowledge of the inter-agent *procedures* used by the monitored organization to maintain the relationships among its members, and to recover in case of failures. This knowledge is used to predict decisions made by agents about their

chosen execution paths, and to predict that certain stages in execution have not been reached (since observations associated with them have not been made).

9.1 Three areas of uncertainty in monitoring teams

To examine the uncertainty and the factors leading to it formally, let us define the notion of *plan-horizon*, which will be useful in grounding the analysis of the areas of uncertainty.

Definition 7. The single-root plan-horizon $SH(A, P, T)$ of an agent A, plan P, and a given elapsed time interval T, is a directed graph in which the vertices are all plans that the agent A may have begun executing during the interval T, if it began with the execution of P.

This includes P, any plan reachable from P by following temporal and hierarchical decomposition transitions, and any plan reachable from any ancestor of P by following first a temporal transition, and then any number of temporal or hierarchical decomposition transitions. The edges in the plan-horizon correspond to the transitions taken. P is called the root of the plan-horizon.

Intuitively, the single-root plan-horizon contains the part of the team-oriented program that corresponds to the hierarchical plan which contains P, and all the hierarchical plans that could temporally follow this hierarchical plan within the time interval T. For instance, given the team-oriented program in Figure 8.1-b, suppose the root of a plan-horizon is the plan `Fly-Flight-Plan`. Then the plan horizon will include `Fly-Flight-Plan`, `Fly-Control-Route`, `Execute-Mission`, `Evacuate`, `Landing-Zone-Maneuvers`, etc.—everything but `Process-Orders` and its children, or other plans that temporally precede `Execute-Mission` or `Fly-Flight-Plan`.

Of course, our observations of A do not necessarily allow us to determine with certainty what was its last known state. For instance, if the last observed message indicated that an agent A has terminated some plan Q, which may be followed by any one of three different plans P_1, P_2, and P_3, then a correct description of the

set of hypotheses about the state of the agent should include three single-root plan-horizons: $SH(A, P_1.T)$, $SH(A, P_2, T)$, and $SH(A, P_3, T)$, where T is the duration of time from the time this message was received.

We thus define the complete plan-horizon of an agent as follows:

Definition 8. The complete plan-horizon $H(A, \{P_1, P_2, \cdots, P_k\}, T)$ (*horizon* for short) is the graph union of the single-root plan-horizons

$$\bigcup_{i=1}^{k} SH(A, P_i, T)$$

where A is a monitored agent, T an elapsed time interval, and P_1,...,P_k the possible states of A when the last observation was made. The set $P = P_1, P_2, \ldots, P_k$ is called the *set of roots* of H.

The number of hypotheses contained in a plan-horizon grows along two dimensions. First, along the temporal dimension, uncertainty about how much time each plan takes to execute results in uncertainty about how many plans have possibly been executed in the time interval defining the horizon. We call this *temporal uncertainty*. The earlier M construct (used in Part I) did not allow for projections involving time, since it was assumed that hypotheses as to the current state can always be generated based on up-to-date observations. Thus in Part I, there was no need to consider temporal uncertainty.

Another factor is the number of alternative plans that an agent may choose from when terminating a given plan in the sequence, or when decomposing a currently-executing plan into child plans (*plan-step uncertainty*). The more uncertain we are about the decisions made by the agent, the more plans we have to include in the plan-horizon, to account for the possible choices made by the agent[1].

The techniques we will explore below alleviate uncertainty in both of these areas. If the monitoring system is able to reduce the temporal and/or plan-step uncertainty of an agent's plan-horizon, then the number of hypotheses considered (whether probabilistically or not) will be reduced. Such reduction is done by using knowledge about what plans are reachable from the roots of the horizon. Transitions that can

[1]The height (depth) of the hierarchies also affects the number of hypotheses. This is independent of the temporal and plan-step uncertainty sources, and we ignore it here for simplicity.

be marked as illegal for a monitored agent to take will reduce these factors and therefore reduce the uncertainty stemming from these two sources.

A third area of uncertainty is fueled by the combinations of the plan-horizons of the team-members, just as in Part I uncertainty in team-modeling hypotheses was fueled by uncertainty in individual-modeling hypotheses (as captured by the M notation). As described in Section 3.2, there is a multiplicative effect when combining individual-modeling hypotheses into team-modeling hypotheses, resulting in an exponential (in the number of agents) number of team-modeling hypotheses.

Overall, the number of hypotheses is unfortunately very prohibitive of practical applications, since the number of hypotheses grows very quickly with the number of agents, the temporal uncertainty of the agents' horizons, and their plan-step uncertainty. The next sections will describe methods of attacking these areas of difficulty by socially-attentive means—disambiguation methods based on knowledge of the relationships that are maintained in the monitored team, and the procedures that maintain them.

9.2 Using knowledge of relationships

Under the assumption that violations of relationships in a team are relatively rare occurrences, one may use the minimal-number-of-failures heuristics to prefer team-state hypotheses that indicate less or no violations of maintained relationships, over those that indicate that no violations have occurred (see Part I of this dissertation). Specifically, we can use minimal-number-of-failures heuristics for the the role-coordination relationships (i.e., assume that an agent will follow its role) and collaboration relationships (i.e., coherence, introduced in Section 3.3).

These heuristics rely on the monitoring system's knowledge of the relationships that are maintained by the team being monitored (Section 3.1). The role-coordination relationships rely on knowledge of the team-hierarchy (who are the members of different sub-teams, what sub-teams are part of what super-teams, what are the different individual roles), and on knowledge of what plans should be executed by what sub-teams/roles. Similarly, the coherence heuristic, introduced in Section 3.3, prefers hypotheses that indicate a minimal number of teamwork failures, and can be used to prefer hypotheses in which members and sub-teams of

the organization are collaborating correctly. To apply the coherence heuristic, we must have a model of the team's execution of the task—allowing us to differentiate the planned team-plans (which require coordination and joint execution) from those plans which the agents are free to execute independently from others, and in which coherence does not apply. Models are readily available in many cases (for instance, in monitoring a team for visualization by its designers and operators). In addition, such models can often be learned in sufficient detail to be useful.

9.2.1 Using Teamwork: Revisiting Coherence

Coherence holds at multiple levels in a team—agents in a *simple* sub-team work together on the plans selected for the sub-team, sub-teams work together with sibling sub-teams on goals joint to the encompassing team, etc. For example, when a *terminate-plan* message is overheard, the recognition system could prefer (because of assumed coherence in the sub-team) the hypothesis that the team-members have received the message, and terminated the plan jointly with the sender. Incoherent hypotheses based on the same overheard message may assume that communication failures have occurred and therefore that some of the other agents have not terminated the plan.

Similarly, but at another level of the team-hierarchy, suppose that the entire flight team (FLIGHT-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`. One hypothesis is that the ESCORT team remained coordinated with the TRANSPORT team, jointly (but quietly) selecting `Landing-Zone-Maneuvers`, and in service of it, `Escort-Ops`. Another hypothesis is that the ESCORT team is still executing `Fly-Flight-Plan`. Yet another is that all but one member of the ESCORT team have selected `Escort-Ops`, etc. The most coherent hypothesis is the one in which the ESCORT team remained coordinated.

Coherence is a very strong constraint, since there is in general only a linear number (in the size of the plan-horizon) of perfectly-coherent hypotheses (compare to the exponential number of incoherent hypotheses). In each sub-team, and at each sub-team level (i.e., sub-team of a sub-team, etc.) the plans that are joint to

the sub-team are shared among all the sub-team-members. Reasoning only about coherent hypotheses (with an assumption of non-failure circumstances) *eliminates,* for these members, the multiplicative effect discussed in Section 9.1).

The coherence heuristic can also reduce the temporal and plan-step uncertainties. Coherence implies that when considering the reachable team-plans for an agent's plan-horizon, the monitoring system can ignore any transitions which are not possible for all the team-members of the sub-team whose plans are being considered. For instance, suppose a transition from a team plan is to be taken only by the TRANSPORT team. Under non-failure circumstances, there are only two coherent hypotheses considering this transition: Either all members of TRANSPORT took the transition, or that none did. If evidence for one member exists supports one of these hypotheses, that evidence can be used to infer the state of the other members.

Formally, this allows us to bound (from above) the number of team-state hypotheses to the minimum of the sizes of the individual agent plan-horizons. This is established in the following theorem:

Theorem 7. *Suppose we have k agents, each with its associated plan-horizon H_i, where $1 \leq i \leq k$. The number of coherent team-state hypotheses when combining H_i is no greater than* $\min\{size(H_i)\}$*, where $size(H_i)$ is the number of hypotheses in the i'th agent plan-horizon H_i.*

Proof. Provided in Appendix B. □

The significance of this property of coherence is that if the monitoring system can reduce the number of individual-state hypotheses for even one agent (by reducing the temporal or plan-step uncertainty of its plan-horizon—see next section), then this reduction will be amplified through the use of the coherence heuristic to apply to the other agents as well.

The use of the coherence heuristic can thus lead to a significant boost in monitoring accuracy, since the number of hypotheses underlying any further (probabilistic) disambiguation is cut down dramatically. Section 9.4 provides an in-depth evaluation of the use of coherence in ranking plan recognition hypotheses in Eavesdropper.

9.2.2 Using the team-hierarchy and knowledge of roles

Another useful heuristic relies on knowledge of the team-hierarchy and roles of the agents involved. This includes knowledge of the agents that are members of a sub-team, the team-hierarchy tree (Figure I-b), and the individual roles of agents within their sub-teams. This knowledge is often readily available in large organizations, such as business enterprises, factories, etc. In our own system it is part of the Team-Oriented Program.

The heuristic assumes a minimal number of failures has occurred with respect to the team-hierarchy relationships, i.e., that agents are not violating their roles within the sub-teams, are not defecting from one sub-team to the other, etc. This can lead to a reduction in the plan-step uncertainty of the agent's plan-horizon, since some alternatives are ruled out (or ranked probabilistically lower) by the monitoring system. For instance, in the evacuation rehearsal domain only one agent (the Quickset agent) is responsible for obtaining the mission orders, while the other agents go into a `Wait` plan. When Eavesdropper considers the possible plans taken by the agents, it rules out the `Wait` plan for the Quickset agent, but considers it alone for the other agents in the team.

9.3 Predicting Team Responses

Section 9.1 discussed three sources of uncertainty in identifying the state of a given individual agent. A reduction of uncertainty in two of these, the temporal and plan-step uncertainty sources, will lead to a reduction in the number of individual-state hypotheses associated with the agent, which in turn will lead to a reduction in the number of team-state hypotheses.

The temporal and plan-step uncertainties are both a product of the plans which are reachable from the roots of the plan-horizon, as previously described. By eliminating transitions (or ranking them probabilistically lower), we decrease the set of plans that are reachable from the roots, and so reduce the temporal and plan-step uncertainties. Of course, this is what the temporal model we have experimented with (Chapter 8) attempted to do: By using a model of average plan durations, it determines the likelihood that a transition was taken, and was thus able to reduce

the temporal uncertainty of the plan-horizon in some cases. However, we quickly discovered that it was insufficient by itself since the variance in execution time of certain plans is very large, for several reasons:

- Plan execution times vary depending on the task execution context. For instance, the traveling plans takes anywhere from 15 seconds to almost two minutes to execute, depending on the particular route being followed.
- Plan execution times vary depending on the execution environment. For instance, when all the agents in the team are running on a local network, their response times to queries may be shorter than when communicating across continents.
- Plan execution times vary depending on the outcome. For instance, when the route-planner is functioning correctly, it responds within a few seconds. However, when it crashes it does not return an answer at all, and the other agents wait for a relatively long time before relying on a time-out to decide that it had failed.

A temporal model based on average plan duration is unfit to handle such variance. Moreover, constructing a sufficient temporal model (i.e., a model that can take into account the context the task, the operating environment, the possible outcome probabilities, etc.) is difficult. Not only would such a model be much more complex, but some its predictions are inherently difficult to calculate. For instance, latency times in the Internet vary greatly, and are difficult to predict.

Fortunately, knowledge of the procedures which the monitored team employs in maintaining its relationships can be used to complement any temporal models, to significantly boost monitoring accuracy. Such knowledge allows the monitoring system to more accurately predict the success/failure outcome of plans, and to predict the timing of future communications, significantly reducing the plan-step and temporal uncertainties, respectively, of the agent's plan-horizon.

9.3.1 Using predicted communications

A team's communication procedures determine at what points during the execution of the task the team will communicate. For instance, the team's communication

procedure may require its members to explicitly communicate when initiating a particular team-plan whose coordinated execution is critical. The procedures may be simple, per-case rules, or may be complex algorithms that take coordination and communication costs into account when arriving at a decision of whether to communicate. For instance, the STEAM teamwork engine (Tambe, 1997) uses decision theoretic procedures to decide on communications based on the cost of the communication versus the projected cost of loss of coordination.

Regardless of the complexity of the procedures, a monitoring system that can gain knowledge of their decision outcomes (the points at which the team will communicate) can significantly boost its monitoring accuracy. The key idea is that given predictions of the communication decisions, the monitoring system can rule out certain hypotheses—hypotheses which involve states that mandated observed messages which were not yet observed by the monitoring system.

Predictions of the communication decisions of the monitored team effectively reduce the temporal uncertainty of the plan-horizons for communicating agents. For instance, consider the TEAMCORE proxies, which communicate to either establish or terminate a plan, i.e., communicate when transitioning from one plan into the next. A correct prediction that a transition will not be taken without a message being intercepted can be used to rule out all hypotheses involving plans beyond the transition in question.

For example, suppose Eavesdropper overhears a message indicating that the flight team has initiated joint execution of `Fly-Flight-Plan` (Figure 8.1). After some time has passed, it is now possible that the team is either still executing `Fly-Flight-Plan`, or it has terminated it already and begun joint execution of `Landing-Zone-Maneuvers`; After some more time has passed, it is possible now that execution has moved on to plans beyond `Landing-Zone-Maneuvers`. However, if the monitoring system knows that the team will explicitly communicate about initiating `Landing-Zone-Maneuvers`, it can eliminate (or rank much lower) the possibility that the team is executing the latter, eliminating any temporal uncertainty of the plan-horizon in this case (only `Fly-Flight-Plan` is possible).

The knowledge required for such team 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. The key idea

in the latter approach 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. We focus on the learning approach first.

A monitoring system may use learning to build a model of the communications that are generated by a team when it works on a particular task. Indeed, we have found that simple rote-learning proved effective in generating a useful communications model that significantly reduced the temporal uncertainty of the plan-horizon in question. This simple mechanism simply records during execution which plans are explicitly communicated about, and whether they were initiated or terminated. The learned rules are effective immediately, and may be stored for future monitoring of the same task.

Figure 9.1 presents the results of learning to predict the communication decisions taken by the team, without the use of any temporal model. The X-axis denotes observed communication message-exchanges as the mission progresses. Overall, some 45 exchanges take place, each one including between one and a dozen broadcast messages in which agents announce termination or initiation of a plan. The Y-Axis shows the number of hypotheses in the corresponding plan-horizon, whose root set is based on each message. Greater temporal uncertainty leads to higher values here.

We see in Figure 9.1 that without learning (the line marked *No Learning*), a relatively high level of ambiguity exists, since the system cannot make any predictions about future states of the agents, other than that they are possible: the plan-horizon thus includes all future states from the current set of roots. However, the size of the plan-horizon is reduced as more observations are made, and past states are ruled out. When the learning technique is applied on-line, some learned experience is immediately useful, and ambiguity is reduced somewhat (the line marked *On-Line Learning*). However, some exchanges are encountered late during task execution, and cannot be effectively used to reduce the ambiguity while learning. The third line (*After Learning*) corresponds to the results when the model has been fully learned. As can be seen, it shows a significantly reduction in the size of the plan-horizon. Further evaluation of the use of communication decisions prediction is presented in Section 9.4. Although this simple learning proved very effective in our experiments, we caution against using it blindly in other tasks and domains, in which the

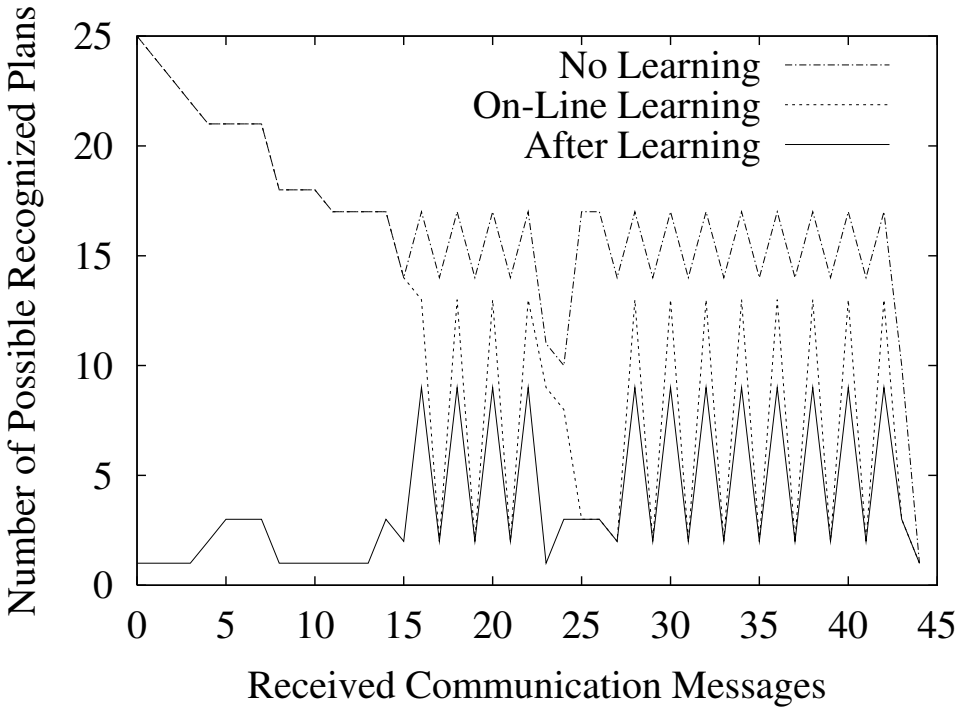


Figure 9.1: Learning of Communication Decisions

communication decisions taken by the team can be more complex. We leave future investigation of more generally applicable learning methods to future work.

9.3.2 Predicting team-responses to failure

Although quite effective at learning the communication responses of the team in the evacuation rehearsal task, the learning approach is problematic for use in predicting other types of responses generated by the team, specifically, responses to failures of agents. Such failures are relatively rare occurrences, and so team responses are potentially difficult to learn due to lack of available data.

Fortunately, the knowledge required for such team responses predictions may be acquired by projecting the state of the team into the future, as previously mentioned. The key idea here is to dynamically generate predictions of the responses based on the known state of the team, and the coordination models used by the team. 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.

These predictions can be useful in several ways. First, they can be used to provide estimates of the likelihoods of failure. For instance, if it is known that in order for the team to successfully terminate the LANDING-ZONE-MANEUVERS plan, at least one member of the *Fly-Out* team must survive, then the probability of the team failing can be calculated based on how many helicopters are known to be crashed, and the probability of a single helicopter crashing. Such a calculation involves an assumption of independence on helicopter crashes (the probability of a helicopter crashing is independent of other helicopters having crashed), but is preferable to the uncertainty resulting from relative lack of data on such complete failures on the part of the team.

A second way in which the predictions can be useful is in predicting what the team will attempt to do when a failure exists. For instance, if the coordination relationship among the helicopters is such that a successful termination depends on the leader of the helicopter team, then if the lead helicopter crashes, we can expect the team to attempt to compensate by selecting a new lead helicopter. Knowledge of the team's responses to failures—the procedures which the team employs to maintain its coordination—can be be captured in team *fault-models* (Horling et al., 1999).

Using such fault-models, Eavesdropper predicts the procedures that a team will execute to handle recovery. For instance, Eavesdropper will predict an exchange of messages that selects the new lead helicopter, and may be able to predict which agent will become the new lead. These predictions allow Eavesdropper to eliminate the "normal course-of-action" transitions from consideration while the the team is recovering.

9.4 Accuracy Evaluation

The emerging solution is thus to construct an array of single-agent recognizers, and use the knowledge of the team's relationships and responses to alleviate the uncertainty. This approach has been implemented in Eavesdropper, which we have been using in actual runs of the system over the Internet. This section evaluates the

contribution of the different techniques in Eavesdropper to recognizing the correct state of the agents and teams.

Figure 9.2 compares the average accuracy for a sample of our actual system deployment runs. The ten runs are marked 'A' through 'J' (X-axis). In each such run, the team executed a complete evacuation simulation mission. At different points during the 10–20 minute execution, the *actual* system state was compared to the state *predicted* by the monitoring system, where that 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 (Y-axis).

The average accuracy when using the individual models with no coherence (Section C) is presented in the leftmost bar (marked *Temporal*) in each group (Figure 9.2). The temporal model used was generated by looking at the average duration of plans in run C. The next bar presents the average recognition accuracy if only coherence is used to rule out hypotheses (Section 9.2), and then an arbitrary selection is made among the remaining hypotheses. The next bar to the right (*Coherent, Temporal*) presents the results of combining both coherence and the probabilistic temporal-reasoning capabilities. First, non-coherent hypotheses are thrown away. Then, the temporal model is used to select among the coherent hypotheses.

The next bar to the right (*Coherent, Comm*) presents the results of combining coherence with the team communication responses predictions described in Section 9.3. In this combined technique, coherence is used in combining the individual plan-horizons. Then the communication predictions are used to decrease the temporal uncertainty of the coherent plan-horizon. Finally, an arbitrary selection is made among the remaining coherent hypotheses. The communications responses predictions were learned based on run C, then modified slightly manually to accommodate different runs, i.e., to prevent the communication model from blocking a transition from being taken. Such modifications relax the effects of the communication models, and in some cases hurt the accuracy of the technique.

The remaining bar (*Coherent, Temporal, Comm*) presents the average accuracy in each run using the full combination of disambiguation capabilities: Coherence, the duration probabilities, as well as the team communication model, which predicts when communications will occur.

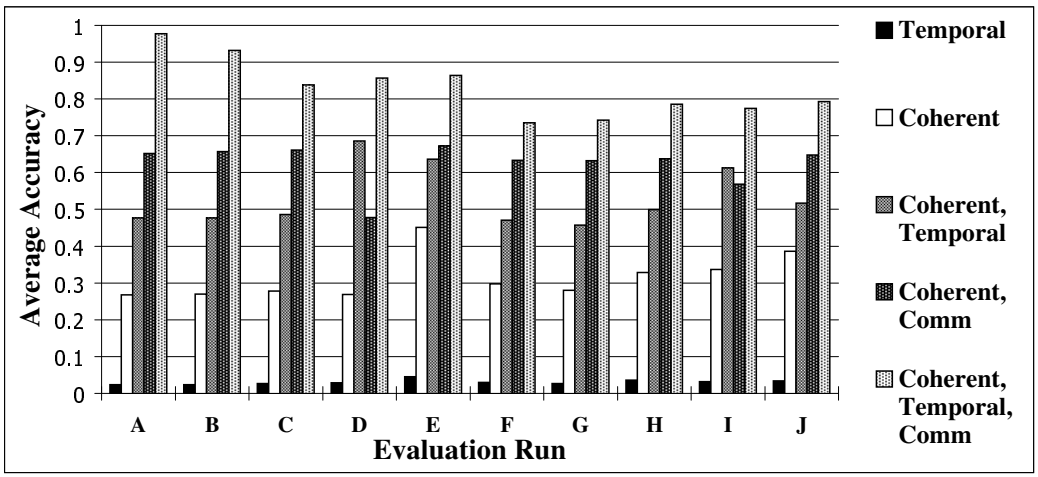


Figure 9.2: Average accuracy in sample runs.

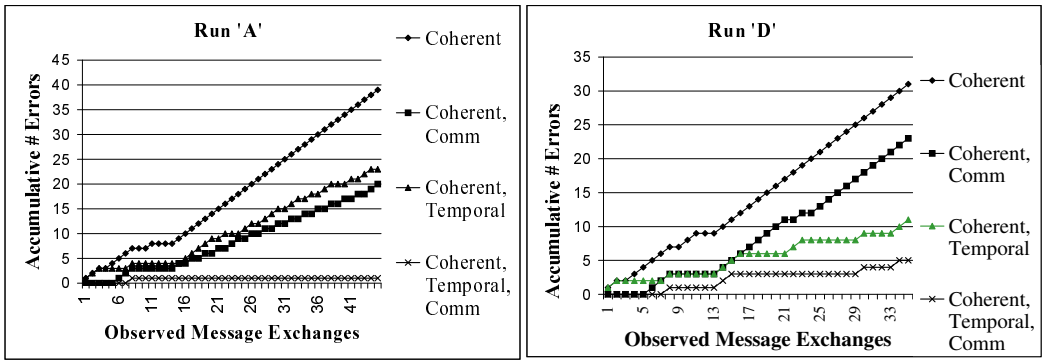
The results presented in Figure 9.2 demonstrate the effectiveness of the socially-attentive monitoring techniques presented. First, the result demonstrate that the coherence heuristic is clearly effective, as it brings the accuracy up by 15–30% without using any probabilistic reasoning. When combined with a temporal model of plan-duration, the average accuracy is significantly increased further, demonstrating that the two techniques are complementary. Indeed, this point is further demonstrated by noting that the right-most bar, showing the average accuracy in each run when combining all techniques is significantly higher than with either technique or simpler combination alone.

A key issue is raised by a careful look at the comparison between the (*Coherent, Temporal*) and (*Coherent, Comm*) bars. These two bars allow us to compare the effectiveness of the temporal model with that of the communication predictions, which we have presented as a solution to the weaknesses of the temporal model. In almost all runs the average accuracy when using coherence and communications predictions is significantly higher than when using coherence and the temporal model. This is despite the fact that the more effective coherence technique uses arbitrary (random) selection among the available hypotheses: The reason for this is that in many cases the communication predictions are powerful enough to rule out all hypotheses but one or two, significantly decreasing the temporal uncertainty of the agents' plan-horizons. Thus even a random selection stands a better chance than a more informed (by a temporal model) selection among many more (10–20) hypotheses.

However, runs D and I show a reversal of this trend. These runs involved relatively more failures on the part of team-members, including agents crashing or not responding at all. The communication predictions, however, were learned based on successful runs—and thus did not correctly predict the communication messages that would result as the team detected and recovered from the failures. Thus the temporal uncertainty of the plan-horizons was not shortened, and the arbitrary selection was made among relatively many hypotheses. This explains the relatively lower accuracy of the (*Coherent, Comm)* technique in run D and I. However, there are other factors that influence the accuracy of the communication models, since this lower accuracy did not occur in other runs where failures have occurred.

The results of the *Coherent, Temporal* technique vary as well. We have been able to determine that failures cause a relative increase in the relative accuracy of the *Coherent, Temporal* technique. However, variance in the results is due to additional factors. In run E, for instance, this technique results in relatively higher accuracy, but no failure has occurred. Certainly, the mission specifications themselves differ between runs, machine loads cause the mission execution to run slower or faster, etc. The great variance in the temporal behavior of the system was the principle reason for our using the communication prediction. This variance is obvious in the graphs.

Figures 9.3a-b provide further evidence for this phenomena. The figure shows the accumulative number of errors as task execution progresses during run A (Figure 9.3a) and during run D (Figure 9.3b). 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. In the two figures, a lower slope means better performance (less errors). 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 using both coherence and the temporal model to choose the most likely hypothesis. Similarly, the line marked *Coherent, Comm* shows the results using both coherence and the team communication responses predictions. Finally, the remaining line displays the results of using coherence, the temporal model, and the model of team communications-responses predictions.



(a) (b)

Figure 9.3: Accumulative number of errors in runs 'A' and 'D'.

In Figure 9.3a, we see that the two techniques (*Coherent, Temporal* and *Coherent, Comm*) have almost equal slopes and result in almost equal number of errors at the end of run A, though from Figure 9.2 we know that due to the much shorter plan-horizon temporal uncertainty, the use of communication predictions leads to overall higher probability of success (i.e., the *Coherent, Comm* technique results in fewer alternative hypotheses, and thus has a better chance of being correct). However, in Figure 9.3b we see that in run D the situation has changed dramatically. First, we see that the two lines are no longer similar. The line marked *Coherent, Comm* has greater slope than in run A, indicating that the communication predictions are not able to reduce the temporal uncertainty of the plan horizon, resulting in lower average accuracy. Second, we see that the temporal model results in many less errors, as evidenced by the much slower-rising slope of the line marked *Coherent, Temporal.* Thus in this case, the actual duration of plans matched the temporal model more accurately than in other runs.

In summary, despite the variance in the results of the *Coherent, Temporal* technique (due to variance in the temporal behavior of the system), and the possible sensitivity of the *Coherent, Comm* technique to learned predictions, it is clear that the two techniques work well in combination, building on the coherence heuristic, and compensating for each other's weaknesses. Together, the coherence heuristic, the communication predictions, and the available temporal knowledge result in very significant increases in accuracy compared to the initial solution with which we began

our investigation (72-97% versus less than 4%). In all runs, the combined technique *Coherent, Temporal*, *Comm* was superior to either technique alone. The communication predictions need to not be perfect, and the temporal knowledge need not be precise, in order to be useful.

Chapter 10

Scaling-Up Monitoring Efficiency

The previous chapter has demonstrated that the uncertainty issue in the monitoring selectivity problem can be successfully tackled by socially-attentive means. However the issue of computational complexity of the monitoring process is also of great concern. The space and time complexities in the initial monitoring solution presented in chapter 8 grew linearly with the number of agents, and thus do not scale well to large-scale teams.

This chapter presents a monitoring approach that trades monitoring expressivity for efficiency— monitoring algorithms that can only represent hypotheses in which the agents are not violating their relationships, but achieve significantly better computational efficiency. Taken to extreme, the solution we provide is able to monitor all agents in a team, regardless of their number and their parallel activities, using a single plan hierarchy. However, it can only reason about coherent hypotheses in which agents do not violate their roles (it may still detect incoherences and role violations). It thus offers a particularly scalable monitoring solution.

10.1 Optimizing using the team-hierarchy

The first step towards this goal begins with the array of single-agent recognizers. In Section 9.2 we have seen that eliminating (or ranking lower) hypotheses in which agents violated their role relationships can reduce the breadth of their respective plan-horizons. We can carry this further to completely eliminate from consideration any such hypotheses—by removing plans and transitions from the plan hierarchy that is used to monitor an agent. In other words, the array-of-recognizers can be

optimized such that the recognizer for each agent uses only the portion of the plan hierarchy that the agent is actually allowed to execute according to the sub-teams of which it is a member, and its roles in those sub-teams.

For example, Figure 10.1 illustrates such an optimized array of single-agent recognizers for the evacuation rehearsal domain. The figure illustrates how the actual plan hierarchy used by each single-agent recognizer changes from agent to agent. The left-most hierarchy, for instance, is unique to the Quickset agent, which is responsible for obtaining the orders for the Task-Force team. the right-most hierarchies shows a portion of the hierarchy used by members of the escort sub-team.

The computational savings in space and time are traded here for expressivity. This optimization physically restricts the system to reason only about hypotheses in which agents do not violate their roles. Hypotheses in which agents fail (by executing incorrect roles, or accidently "defecting" to another sub-team) cannot be explicitly represented. This may prevent the monitoring system from hypothesizing failures without direct evidence.

Role/Sub-team violations can still be detected, if communications are observed that unambiguously indicate that an agent is executing plans that it is not supposed to (based on its role or sub-team). The optimization precludes some plans from appearing in the plan hierarchy of a given agent, i.e., its plan-horizon will be restricted to those plans that it may actually execute. If a message is received indicating that the agent has begun executing a plan that is not in the plan-horizon, then this is a clear indication that the agent is violating the known role/sub-team relationships (or that our model of the team-hierarchy is incorrect). Although the monitoring system will be able to detect this failure, it will not be able hypothesize the erroneous state of the agent, i.e., the monitoring system will know that the agent has violated the team-hierarchy, but not how. Section 10.3 presents an evaluation of this expressivity-efficiency trade-off.

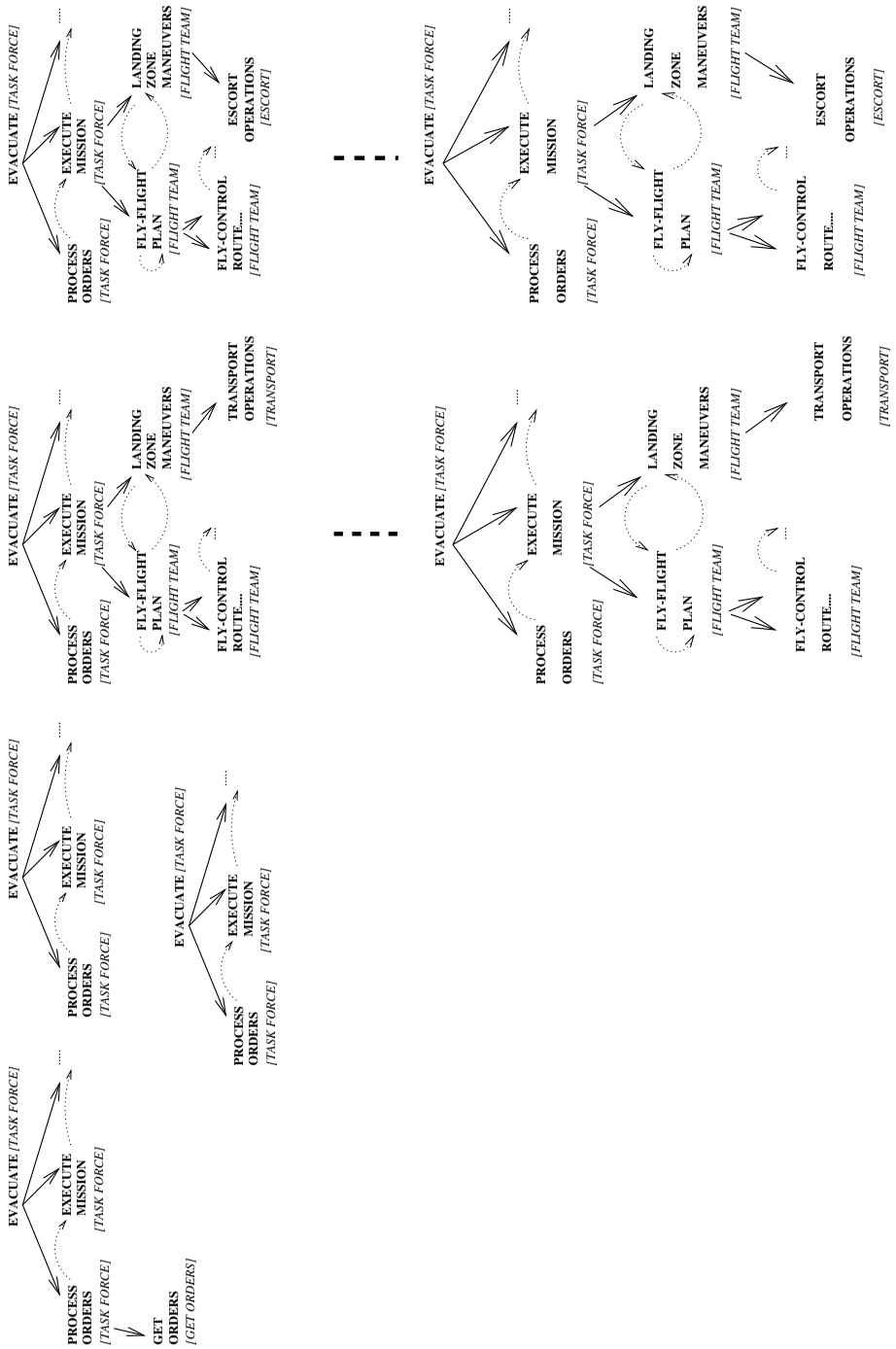


Figure 10.1: Array of single-agent recognizers, optimized (the plan-library changes based on agent's role).

10.2 YOYO*: Efficiently reasoning about coherent hypotheses

It is clear (Section 9.4) that coherence is an extremely important heuristic to increased monitoring accuracy. However, the monitoring algorithm that uses an array of single-agent recognizer is not optimized for representing coherent hypotheses. Each agent's state is represented separately, and coherence is enforced as a post-processing constraint rather than a-priori. A more extreme step towards improved efficiency (by sacrificing expressivity) can be taken by restricting the reasoning to a-priori rule-out incoherent hypotheses.

We present here the YOYO* family of algorithms, which optimize monitoring by restricting reasoning to coherent hypotheses only. At the expense of this decreased expressivity, YOYO* algorithms use only a single plan hierarchy to represent any number of agents in a team. This provides a particularly scalable solution for large teams, as any reasoning (such as propagating probabilities) can take place once for the entire team.

The key idea is to keep the plan-horizons of all agents coherent, such that the plan-horizons of different agents do not imply any incoherences. YOYO* maintains a team-hierarchy (such as the one presented in Figure 8.1), such that each agent *and each sub-team* (i.e., every node in the team-hierarchy tree) has associated with it a set of pointers to the roots of their current hypothesized plan-horizons in a *shared plan hierarchy*.

These pointers are maintained coherent with each other at all times. When an observation is made about an agent (called the *focus*), we not only update the pointer for this agent, but also re-align the pointers of its parent and child teams, such that their own pointers point at a set of roots that is coherent with the new roots of the focus. We then go up and down the team-hierarchy to re-align the pointers of the other agents which are either part of the focus' sub-team or its siblings[1]. This re-alignment is done by moving the roots of the non-focus agents (and the sub-teams of which they are members) forward (in their horizons) until they point at a root that

[1]This up and down traversal of the plan- and team-hierarchies is the root of the name for this family of algorithms.

is coherent with the plan-horizon of the focus. If the initial set of roots is already coherent with the focus' new root, no re-alignment is necessary.

To illustrate, let us re-examine the following example. In our application, the entire team (Task-Force) jointly executes the `Execute-Mission` plan (Figure 8.1). In service of this plan, Task-Force may either execute the `Fly-Flight-Plan` plan or the `Landing-Zone-Maneuvers` plan. In service of the latter plan, the TRANSPORT team must execute the `Transport-Ops` plan, while the ESCORT team executes the `Escort-Ops` plan.

Suppose that the entire team, based on previous messages, is known to be executing the `Fly-Flight-Plan` plan (i.e., the roots for Task-Force, ESCORT and TRANSPORT—and their members—are all `Fly-Flight-Plan`). Then a message is received that indicates that the transports are now executing the `Transport-Ops` plan. YOYO* then updates the root of the TRANSPORT to be the `Transport-Ops`. Under the assumption that the team is coherent, it then infers (climbing up the team-hierarchy) that the TRANSPORT team has terminated `Fly-Flight-Plan` jointly with the other sub-teams, i.e. that the parent team Task-Force has terminated `Fly-Flight-Plan` and has begun executing `Landing-Zone-Maneuvers`. It therefore updates the root of Task-Force to be `Landing-Zone-Maneuvers`. Furthermore, it now infers (climbing back down the team-hierarchy) that the escorts team is executing `Escort-Ops` in service of `Landing-Zone-Maneuvers`, and so it updates the root for the ESCORT team to be `Escort-Ops`.

A symbolic version of YOYO* (which does not propagate probabilities) is presented in Algorithm 10.1 (a probabilistic version is presented in Appendix D). It uses a single fully-expanded plan hierarchy M, which is the graph-union of all the agents' plan hierarchies, i.e., all possible plans and transitions for all roles. It is assumed that M is annotated such that YOYO* can tell what transitions and plans are allowed to be executed by which teams. YOYO* also relies on an extended model of the team hierarchy H, which contains a pointer for every team into M. These pointers point at the roots of the teams' current plan-horizon.

When a monitoring query as to the state of a particular agent or sub-team is made, the monitoring system simply looks up the agent or sub-team in the team-hierarchy. It then uses the pointers to find out the roots for the agent or sub-team, and computes the plan-horizon dynamically (using the time of the query) to answer

the query. The dynamic computation of the plan-horizon uses the sub-team and role tags in M to make sure only transitions and plans executable by the query sub-team are considered.

Algorithm 10.1 YOYO*(**plan hierarchy** M, **team-hierarchy** H)

```
1: Loop forever:
2: if messages received–new plan S team T,  then
3:    locate T in the team-hierarchy tree H
4:    set the roots of the plan-horizon of T to all S plans in T's current plan-horizon
5:    set the roots of the plan-horizon of any child sub-team of T to S
6:    tmp ← T
7:    while tmp is not the root team in H do
8:       find lowest common ancestor N of S shared by tmp and its sibling teams
9:       if N different than previous plan shared with the siblings then
10:         set the roots of parent and siblings of tmp (and their children) to N
11:      tmp ←parent_ of (tmp)
```

To illustrate the operation of YOYO*, consider how it may handle the example of the transports and escorts as described above. Suppose that the entire team is known to be executing `Fly-Flight-Plan`: For any sub-team in H, the current state can be found by using the plan-horizon rooted by the pointer to `Fly-Flight-Plan` in M. Now, a message exchange is observed among the members of the TRANSPORT team, indicating that they have initiated execution of `Transport-Ops` (Line 2 in Algorithm 10.1). First, the TRANSPORT team is located in H (Line 3), and its new plan-horizon root `Transport-Ops` is found in M (Line 4). For each sub-team of the transports, we update the pointers it contains in H to now point to this root (Line 5). We now enter the loop (Lines 6-7). The first parent plan of that is shared to the transports and its sibling sub-team, the escorts (Line 8), is `Landing-Zone-Maneuvers`, which is found by climbing from `Transport-Ops`. This shared parent is new–it is different than the previous shared parent `Fly-Flight-Plan`. Therefore, `Landing-Zone-Maneuvers` is now pointed to by the ESCORT team, and the common parent team FLIGHT-TEAM (Line 10). Now we look at the FLIGHT-TEAM (Line 11, returning to Line 7). Its new root is now `Landing-Zone-Maneuvers`. Its new shared parent (with its sibling sub-teams) is `Landing-Zone-Maneuvers`. This is different than the previous parent, and so

we change the sibling and parent teams' roots to `Landing-Zone-Maneuvers`. We continue to the parent team Task-Force, and then the while loop terminates.

10.3 Efficiency Evaluation

We examine here a key trade-off between the expressivity and efficiency involved in the monitoring techniques we have presented in this chapter. 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 the TRANSPORT team is executing `Transport-Ops` in service of `Landing-Zone-Maneuvers`, while the ESCORT team is executing `Process-Orders`. It is thus often impossible to explicitly represent hypotheses associated with communication losses and delays, which cause such incoherences.

However, YOYO* is still able to detect many (even if not all) such incoherences. The incoherences would be detected when a new set of roots (based on new evidence) cannot be found in the plan-horizon for the agent. This happens when the roots for the agent have been previously aligned based on its team-mates or other sub-teams. For instance, the incoherence described above could be detected as follows. Suppose that a message is overhead indicating that one TRANSPORT member has initiated execution of `Landing-Zone-Maneuvers`. As previously discussed, YOYO* would re-align the roots of the TRANSPORT team to `Landing-Zone-Maneuvers`, then re-align the roots of the ESCORT team to the same plan, to maintain coherence. Suppose that another message is overheard now in which a member of the ESCORT team indicates initiation of `Process-Orders`. This plan is not in the plan-horizon for the ESCORT team-members, since the plan does not allow for this new plan to be executed *after* `Landing-Zone-Maneuvers`. Thus, there is no way to identify the new roots for the ESCORT members (and the rest of the team), and the incoherence would be detected. However, without further auxiliary information, YOYO* cannot identify the hypotheses involved in the failure: it cannot identify the plan-horizon rooted at `Process-Orders`. In contrast, the approach employing an array of single-agent recognizers, 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, plan hierarchy M to represent the entire team. This model is a union of all the individual agent task-models, and therefore its space complexity is bounded from above by the combined size of the individual models. In the worst case, in which each agent has its own set of plans and no two agents jointly execute a plan, the size of M will be the same as the combined size of the role-optimized array of single-agent recognizers as in Section 10.1. However, in the best case all the plans will be team-plans which are to be jointly executed. Then, the size of M will be the same as the size of a single-agent plan hierarchy. Table 10.1 below provides empirical results in our domain.

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.

Table 10.1 contrasts the different approaches, highlighting expressivity versus efficiency. The first column notes the monitoring approach discussed, and the section in which it is discussed. The second column provides the space complexity of the approach. Here, N_A is the number of agents, and R is the number of different roles in the team ($1 \leq R \leq N_A$). m is the constant size of a fully-expanded plan hierarchy (i.e., a plan hierarchy that contains information for all roles and sub-teams). The next column provides the actual number of plans (not including transitions) used when implementing the approach in our domain. The last column provides a summary of the expressivity of the approach in terms of hypotheses that it can and cannot represent.

We see that the full array approach has both a worst- and best- case space complexity of $O(N_A * m)$. This is because we use a full plan hierarchy to represent each of the agents. It is expensive (726 plans are maintained, with additional 66 for each additional agent), but allows for full representation of all types of hypotheses,

Monitoring Algorithm	Space Complexity	Empirical	Expressivity
Array (Chapter 8)	$N_A * m$	726	Teamwork Failures: Sound or Complete Role Failures: Full expressivity
Optimized Array (Section 10.1)	$N_A * \frac{m}{R}$	496	Teamwork Failures: Sound or Complete Role Failures: Detection only
YOYO* (Section 10.2)	m	66	Teamwork Failures: Sound (detection only) Role Failures: Detection only

Table 10.1: Trade-offs in monitoring efficiency versus expressivity

including hypotheses in which agents have defected from the roles, or incoherences exist within sub-teams. In terms of teamwork failure detection, then, it is able to provide either sound or complete detection (Theorems 1-3).

The optimized array approach cuts down the space requirements significantly, by maintaining for each agent only the portion of the plan hierarchy that is actually allowed for the agent based on its role and sub-team. Its space complexity is $O(N_A * \frac{m}{R})$. In the worst case, all agents have the same role ($R = 1$), and thus no space savings are actualized with this approach compared to the non-optimized array, since the same exact plan-library (of size m) would be used in monitoring each individual agent. In the best case, however, each agent has its own role which does not share any plans with any other agent ($R = N_A$). Then the total size of the array is just m, since the plan-library for each individual agent is only of size $O(\frac{m}{R})$. Our implementation used 496 plans (savings of 32%), with up to 40 additional plans with each additional agent. The approach thus promises better space requirements, but sacrifices the ability to reason about role failure hypotheses[2]. It is still able to reason about incoherences, and is thus able to provide either sound or complete detection as the non-optimized version.

Finally, the YOYO* approach provides the best guarantees on space complexity—$O(m)$—using only 66 plans in the implementation, with no additional plans when adding agents. The savings are of 91% compared to the full array approach, 87% compared to the optimized array approach. However, it sacrifices the capacity for reasoning about incoherent hypotheses, offering only sound detection capabilities in such cases.

[2]However, role-failures may still be detectable.

Chapter 11

Related Work

Our investigation of the monitoring selectivity problem, and the socially-attentive monitoring methods for addressing it, builds on research in different fields: Teamwork and coordination, execution monitoring, diagnosis, plan recognition, human team training, team behavior analysis, and more. We address these fields in this section, and explain how our investigation is related to existing literature.

11.1 Teamwork

Previous work in teamwork has recognized that monitoring other agents is critical to teams. Past investigations have raised the monitoring selectivity problem, but have not addressed it in depth. Building upon these investigations, this dissertation begins to provide some in-depth answers to this problem.

The theory of SharedPlans (Grosz & Kraus, 1996, 1999) touches on the teamwork monitoring selectivity problem in several ways, but provides only some initial answers. First, the theory requires agents to know that their teammates are capable of carrying out their tasks in the team. The authors note that agents must communicate enough about their plans to convince their teammates of their ability to carry out actions (Grosz & Kraus, 1996, p. 314). Second, the theory requires agents to have mutual-belief in the shared recipe, a state that requires agents to reason to infinite recursion about other agent's beliefs. Unfortunately, attainment of mutual belief is undecidable in theory (Halpern & Moses, 1990) and hence must be approximated in practice (Jennings, 1995; Rich & Sidner, 1997). Such approximations may still impose strong monitoring requirements. Third, the theory introduces

the intention-that construct in service of coordination and helpful behavior, implying monitoring of others' progress to assess the the need for such behavior (Grosz & Kraus, 1996, Axiom A5-A7). Fourth, SharedPlans requires that intentions of an agent must not conflict (Grosz & Kraus, 1996, Axiom A1), and since some of these intentions (in particular, intentions-that) may involve the attitudes of other agents, some monitoring of others to detect and avoid conflicts is implied. The authors point out that while theoretically all such conflicts can be detected, this is infeasible in practice (Grosz & Kraus, 1996, p. 307). They suggest that conflict detection and prevention be investigated in a problem-specific manner within the minimal constraints (i.e., monitoring for capabilities, mutual-belief, progress, lack of conflicts) provided by the SharedPlans framework (p. 308 and 314).

Joint-Intentions (Levesque et al., 1990; Cohen & Levesque, 1991) requires an agent who privately comes to believe that a joint-goal is either achieved, unachievable, or irrelevant, must commit to having the entire team mutually believe it to be the case. As in the theory of SharedPlans, Joint-Intentions' use of mutual belief can only be approximated in practice, and imposes strong monitoring requirements. Thus, the monitoring selectivity problem is raised for practical implementations of Joint-Intentions.

Jennings has hypothesized that two central constructs in cooperative multi-agent coordination are *commitments* made by the agents, and *conventions*, rules used to monitor these commitments (Jennings, 1993). Such conventions are used to decide what information needs to be monitored about agents, and how it is to be monitored. For instance, a convention may require an agent to report to its teammates any changes it privately detects with respect to the attain-ability of the team goal. Jennings raises the monitoring selectivity problem and provides an example of specific conventions for high- and low-bandwidth situations in which some knowledge is not communicated to all agents if the bandwidth is not available. However, Jennings does not explore in-depth the question of how such conventions are selected, and what are the trade-offs and guarantees associated with the selection of particular conventions. For instance, there are no guarantees on the effects of using the low-bandwidth convention in the example. In contrast, our work provides analytical guarantees on the monitoring goals which may be achieved using the monitoring techniques we introduced for failure detection.

The theoretical investigations described above all raise the monitoring selectivity problem (implicitly or explicitly). Our work builds upon these to address this problem in depth, in the context of socially-attentive monitoring in teams. This dissertation reports on soundness and/or completeness properties of teamwork relationship failure-detection that can be analytically guaranteed, despite uncertainty in knowledge acquired about monitored agents. The analytical guarantees are applicable to plan recognition and communications, and are corroborated by empirical results.

Building on theoretical work, practical teamwork systems include GRATE* (Jennings, 1995), COLLAGEN (Rich & Sidner, 1997), and STEAM (Tambe, 1997). Jennings' investigation of the Joint-Responsibility teamwork model in GRATE* (Jennings, 1995) builds on Joint-Intentions, and similarly to our own implementation, requires agents to agree on the team-plans which are to execute. However, GRATE* is used in industrial settings in which foolproof communications can be assumed (Jennings, 1995, p. 211), and thus only passive monitoring (via communications) is used. Although Jennings provides an evaluation of GRATE*'s performance with respect to communication delays, no guarantees are provided with respect to failure detection. GRATE* maintains knowledge about other agents through acquaintances models, which are used to keep track of what team-members' capabilities are (in service of forming teams). However, the question of how much knowledge should be used in these models is left unaddressed.

Rich and Sidner investigate COLLAGEN in a collaborative user-interface system, in which communications are reliable (Rich & Sidner, 1997). However, from a human-usability perspective, limiting the amount of communications is still desirable. To address this issue, recent empirical work by Lesh, Sidner and Rich (1999) utilizes plan recognition in COLLAGEN; the focus of that work is on using the collaborative settings to make the plan recognition tractable. For instance, ambiguities in plan recognition may be resolved by asking the user for clarification. Work on COLLAGEN does not investigate how much knowledge is to be maintained for effective collaborative dialogue with the user. In contrast, we are able to provide guarantees on the failure-detection results of our algorithms. Also, analyzing the dialogue plans for *risky points* may allow systems such as COLLAGEN to decide

whether to use communications for clarification even when plan recognition is not ambiguous.

STEAM (Tambe, 1997) maintains limited information about the ability of team-members to carry out their roles. STEAM also allows team-members to reason explicitly about the cost of communication in deciding whether to communicate or not. Our work significantly extends these capabilities via plan recognition, and provides analytically-guaranteed fault-detection results. Furthermore, our teamwork failure-detection capabilities can be useful to trigger STEAM's re-planning capabilities.

11.2 Multi-Agent plan recognition

Eavesdropper and RESL both employ multi-agent plan recognition to monitor agents. 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; Devaney & Ram, 1998)), a key novelty in Eavesdropper is that it effectively blends these two threads together.

$RESC_{team}$ (Tambe, 1996) focuses on explicitly using team intentions for inferring *team plans* from observations, similarly to Eavesdropper's use of the coherence heuristic, and used coherence to restrict the space requirements of the plan-library used, similarly to YOYO*. When used to model adversaries, $RESC_{team}$ employs a worst-cost heuristic which reasons only about the hypotheses which imply the most cost to the monitoring agent. $RESL$'s representation is similar to that of $RESC_{team}$, but $RESL$ allows for explicitly representing multiple hypotheses, instead of one, and for a variety of disambiguation heuristics to be used, instead of coherence alone. Eavesdropper uses a much more advanced team model than $RESC_{team}$ (e.g., it can predict failures based on coordination constraints), uses knowledge about the procedures used to maintain the relationships (i.e., when communications will take place), and also explicitly reasons about uncertainty and time, allowing it to answer queries related to the likelihood of team plans and failures, given a future target time.

van Beek and Cohen (1991) have looked at plan recognition ambiguity in the context of generating cooperative responses. They identified a set of conditions

under which an automated help system used by students actually needs to disambiguate between recognized alternatives of students intentions by communications, and provided inspiration to some of the work reported on here. However, their system represents the ambiguity explicitly in any case, by critiquing and reasoning about the alternatives. In contrast, we have empirically and analytically identified a set of conditions under which the representation itself is not necessary for monitoring purposes. We were able to address the need for monitoring disambiguation in a principled way, based on knowledge of the relationships that hold among monitored agents. In Part I, we provide analytical guarantees on the requirements and results for such selective disambiguation, in service of failure-detection. In Part II, we have addressed situations in which accurate monitoring is required, and shown that socially-attentive monitoring can reduce uncertainty even in these cases.

Washington (1998) presents an approach to plan recognition via a Partially-Observable Markov Decision Process (POMDP). Washington shows that when POMDPs are used to represent a monitored agent's state, reasoning is possible in real-time scales, as long as the monitoring agent does not influence in any way the behavior of the monitored agent. This is an answer to the monitoring selectivity problem when POMDPs are used for representation. Our work differs from Washington's in several ways: First, we do not restrict the monitoring agent from acting in any way; second, the plan recognition representations we present do not share the computational requirements of POMDPs; and third, we provide a set of conditions under which no representation of uncertainty in the plan-hypotheses is necessary for failure detection.

Intille and Bobick (1999) rely entirely on coordination constraints among agents to recognize team-tactics in football, and in this sense use one of our socially-attentive techniques, which prefers hypotheses in which agents are maintaining their roles. Similarly, Devaney and Ram (1998) use pattern matching to recognize team-tactics in military operations. Their approach relies on task- specific pre-defined team-plan libraries, which are verified by domain experts. Both investigations use position trace data of the monitored human teams. Our work differs from these investigations in several ways. First, we take the position that 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, in a case where a team member were to suddenly fail, and its teammates will change their behavior to render assistance or otherwise compensate for its failure. In contrast, Eavesdropper can predict role replacement and continue with its monitoring. Furthermore, in its most general form, Eavesdropper can reason explicitly about teamwork and role failures, which would confuse the previous approaches. Second, these previous investigations have been applied in settings where observations are continuously available about each monitored agent. In contrast, Eavesdropper is targeted towards applications where limited observations are available. It introduces a number of novel techniques (such as the communication predictions) which are significantly useful in such domains.

Huber (1995) investigated the use of probabilistic plan recognition in service of active teamwork monitoring, motivated by the unreliability and costs of passive communications-based monitoring in military applications. In (Huber & Hadley, 1997), Huber reports on the use of probabilistic plan recognition in service of observation-based coordination in the Net-trek domain, and shows that agents using plan recognition for coordination outperform agents using communications for coordination. Huber takes coordination to be cooperative actions on the part of the self- interested agents–for instance, joining an agent in attacking a common enemy. Huber's work does not exploit any knowledge of relationships between the agents to limit the computation or increase the accuracy, and does not address the monitoring selectivity problem, in contrast to our work.

The general probabilistic plan recognition approach of Charniak and Goldman (1993) could explicitly reason about uncertainty in multi-agent systems. The coherence and communication predictions technique we utilize could, in principle, be applied to the general representation they propose. However, the potentially unbounded number of observed communication in the target domains would render the approach impractical for online recognition. In contrast, our work is specifically targeted towards on-line recognition. In particular, our YOYO* algorithm and representation is specifically geared towards efficient recognition.

11.3 Multi-Agent Monitoring Selectivity

Durfee (1995) discusses various methods of reducing the amount of knowledge that agents need to consider in coordinating with others. The methods discussed involve pruning parts of the nested models, using communications to alleviate uncertainty, using hierarchies and abstractions, etc. Our work differs from Durfee's in several important ways: First, Durfee's work focuses on methods by which modeling can be limited. We focus on socially-attentive monitoring methods, and empirically and analytically explore their effectiveness with respect to the monitoring task, providing answers to the monitoring selectivity problem. For instance, we provide analytical guarantees on trade-offs involved in using limited knowledge of agents for failure-detection purposes. Second, we address the monitoring selectivity problems in the context of agent teams, rather than self-interested agents as in Durfee's work, which emphasizes decision-theoretic representation and reasoning. Our focus on teams allows us to bring to bear knowledge of teamwork in service of socially-attentive monitoring methods.

Sugawara and Lesser (1998) report on the use of comparative reasoning/analysis techniques in service of learning and specializing coordination rules for a system in which distributed agents coordinate in diagnosing a faulty network. The investigation is focused on optimizing coordination rules to minimize inefficiency and redundancy in the agent's coordinating messages. Upon detecting sub-optimal coordination (via a fault model), the agents exchange information on their local views of the system and the problem solving activity, and construct a global view. They then compare the local view to the global view to find critical values/attributes which were missing from the local view and therefore gave rise to the sub-optimal performance problem. These values and attributes are used in constructing situation-specific rules that optimize coordination in particular situations. For example, network diagnosis agents may learn a rule that guides them to choose a coordination strategy in which only one agent performs the diagnosis and shares its result with the rest of the diagnosis agents. Our work on socially-attentive monitoring similarly uses comparison between agents views to drive the monitoring process. However, our use of comparison is a product of the relationship we are monitoring. While Sugawara and

Lesser's work can be viewed as letting the agents incrementally optimize their monitoring requirements, our results analytically explore the level of monitoring required for effective failure-detection, in different configurations. Our teamwork monitoring technique addresses uncertainty in the acquired information, and does not construct a global view of all attributes the system—as that would be extremely expensive. Instead, our technique focuses on triggering failure detection via contrasting of plans, then incrementally expanding the search for differences in the diagnosis process.

Klein and Dellarocas (1999) recognize the importance of monitoring relationships in a multi-agent system in service of failure detection and diagnosis. Their work focuses on off-line analysis of various types of coordination relationships resulting in failure models of these relationships. These failure models are then used by centralized monitoring conditions during execution to detect failures. Our technique predates theirs and differs from it in critical ways: (i) In contrast to their work, we provide guarantees on the effectiveness of the techniques, as well as empirical validation; (ii) our technique uses general consistency-based diagnosis which is able to detect failures that have not been anticipated by the designer, while their approach inherently relies on the designer to predict what failures will occur; (iii) we have shown that centralized teamwork monitoring systems are inherently limited, while their work relies heavily on the centralized monitoring (though not necessarily in the context of teamwork); (iv) our investigation explicitly treated issues of uncertainty and the cost of communications and observations, while their technique does not take these issues into account.

Robotics literature has also raised the monitoring selectivity problem. Parker (1993) empirically investigated the monitoring selectivity problem from a different perspective, for a formation-maintenance task. She empirically examined the effects of combining socially-attentive information (which she referred to as local) and knowledge of the team's goals, and concludes that the most fault-tolerant strategy is one where the agents monitor each other as well as progress towards the goals. In contrast, we have explored explicit teamwork and coordination relationships, not formations, and provide analytical guarantees as well as empirical investigation.

ALLIANCE (Parker, 1998) is an architecture specifically built to provide fault-tolerance in cooperative tasks by having the each robot monitor its task's execution, and communicate its monitoring result to the other robots. A mechanism of

acquiescence and impatience allows robots to dynamically give up and take over tasks to compensate for any failure. Kuniyoshi et al. (Kuniyoshi, Rougeaux, Ishii, Kita, Sakane, & Kakikura, 1994) present a framework for cooperation by observations, in which robots visually attend to others as a prerequisite to coordination. The framework presents several standard attentional templates, i.e., who monitors whom. They define a team attentional structure as one in which all agents monitor each other. In contrast to these investigations, our work focuses on the monitoring selectivity problem within socially-attentive monitoring of teamwork relationships, and provides analytical as well as empirical results. We treat the attentional templates as a product of the relationships that hold in the system. In particular, our results show that monitoring in teams may not necessarily require monitoring all team-members.

11.4 Monitoring in Single-Agent Settings

The monitoring selectivity problem has been addressed in the context of single-agent monitoring for discovering opportunities, detecting failures, and visualization. As a result of the single-agent focus, the techniques addressed the monitoring selectivity problem in monitoring the environment, not other agents. In contrast, socially-attentive monitoring focuses on the monitoring selectivity problem in monitoring other agents, complementing the single-agent techniques in multi-agent applications. These socially-attentive monitoring techniques are able to detect previously undetectable failures and opportunities. Additional differences with systems are noted below.

Doyle et al. (1986) describe a method for generating perceptual expectations for verification of plan execution. Their system analyzes a given plan, and inserts appropriate perception requests to verify that the intent of plan steps has been achieved, and that preconditions of next steps hold. Reece and Tate (1994) take a more active approach to monitoring. They present a technique for generating protection monitors, conditions that verify that the world state is as expected in-between plan steps. Recently, Veloso et al. (Veloso et al., 1998) further extended this type of automated monitoring system by introducing Rationale-Based Monitoring, a monitoring technique in which new monitors are inserted to keep track of environmental

features that are associated with planning decisions, not simply precondition values. This allows the planner not only to detect failures in the plan, but also to discover opportunities for optimizing the plan due to changes in the world. In contrast to all three systems, the creation of social monitoring conditions is done dynamically by our system at execution time by using a model of the relationship. Compiling those conditions at design time, for increased efficiency at run-time is left for future work.

Cohen et al. (1992) explored performance envelopes, which measure progress in an action of long duration. Envelopes allow the monitoring agent to find out the degree to which the failure is occurring, how much time the agent has in order to recover from the failure, and whether a failure state is likely in the future. Atkins et al. (1997) focus on the requirement to handle novel, possibly failure, states under hard real-time constraints. They describe a set of techniques based on learning and planning-time characterizations of possible failure states (states for which no plan is available), so that these can be recognized quickly during execution by CIRCA, their plan execution system. In contrast to both investigations, our work in Part I focuses on post-failure detection and recovery. However, with recent work on extending STEAM to handling persistent teams (Tambe & Zhang, 1998), in which STEAM attempts to search for future failure states, we think it is likely that this capability will be easy to integrate in the future. Our work on Eavesdropper (Part II) integrates organizational fault-models into the monitoring system, which are used to predict failure probabilities based on the current state of the agents. While not as systematic an approach as Hart et al., these fault-models do offer some early-warning capabilities.

SELMON (SELective MONitoring) (Doyle, Chien, Fayyad, & Wyatt, 1993; Doyle, 1995) is a real-time monitoring system designed to use multiple failure detection techniques. Besides the condition monitors and model-based diagnosis techniques, SELMON allows detection based on deviation from the known historical performance of a system. In contrast to these features, our work on on-line failure detection relies on known relationships between the state of components' (agents) in the system to identify failures. We are able to provide analytical guarantees on the failure-detection capabilities of our techniques. SELMON additionally uses a model of the historically known causal relationships between sensor readings to focus the human operator's attention on the areas of the system that at the root of

the malfunction, rather than symptomatic. This is somewhat akin to Eavesdropper use of the communication and failure responses predictions to increase monitoring uncertainty. However, while SELMON only attempts to direct the attention to the operator towards relevant sensor readings, Eavesdropper's task is to completely identify the system state.

11.5 Diagnosis

Horling et al. (1999) present an integrated diagnosis system for a multi-agent system in the context of an intelligent home environment. The system uses the TAEMS domain-independent multi-agent task-decomposition and modeling language to describe the ideal behavior of each agent. The agents are also supplied with additional information about the expected behavior of the environment they inhabit under different conditions, and their role within the multi-agent organization. A distributed diagnosis system, made of diagnosis agents that use fault-models, is used to identify failures in components (such as erroneous repeated requests for resources) and inefficiencies (such as over- or under-coordination). The fault-models are used in planning monitoring actions, in identifying failures responsible for multiple symptoms, and in guiding recovery actions. Multiple diagnosis agents may use communications to inform each other of their actions and diagnoses.

Our proposed diagnosis method (Section 3.4) is model-based, and does not use fault-models (also referred to as fault dictionaries (Hamscher et al., 1992)). In other words, we use a model of ideal behavior (in terms of the relationships), not a model of how failure symptoms relate to possible failure diagnoses. The model-based approach has the advantages of generality and model re-use (Hamscher et al., 1992). In particular, fault models, as used by Horling et al., are anticipatory—they are only able to capture failures which the designer has been able to anticipate in advance. Our approach to diagnosing failures is not limited in this respect. Our work on Eavesdropper does integrate fault-models into the monitoring framework, but these are different than the models used in Horling et al's work in two ways: First, our fault-models are used in service of recognizing failure and recovery actions by the monitored team, not in service of actual recovery of the failures by the monitoring

system. In addition, the fault models we use are social in nature—they do not model reasons for failures, but how failures affect the team.

Hudlicka and Lesser (1987) report on a centralized system, DM, which uses problem solving execution traces to diagnose failures. DM uses knowledge of the internal problem-solving system structure as a causal model, and traces of execution to get at intermediate problem- solving states. Once failure is detected, the system uses the causal model and known states to attempt to diagnose the failure. Of most relevance here is the fact the often, there is no clear knowledge that allows the system to decide why a failure had occurred, since there could be many reasons. DM then uses *comparative reasoning*, to compare the problem solving trace that resulted in a failure with a similar problem-solving trace that resulted in success. Differences between the two traces are used to focus the diagnosis on relevant parts. It is thus the similarity of the causal structure between the traces that is used to drive the diagnosis, since the actual ideal behavior is unknown. Socially-attentive monitoring also attempts to use relationships in service of monitoring and diagnosis, but allows for different types of relationships used, and of course has the advantage of working with standard social relationships that exist among multiple agents, instead of the similarity in causal structure that is used by DM. Although in TEAMORE, socially-attentive monitoring is using execution-traces for monitoring and diagnosis, our focus has been on-line, multi-agent observation-based monitoring system. In particular, we discuss benefits of distributing the monitoring task unlike DM which is centralized.

Dressler (1994) distinguishes between inter-state and intra-state diagnosis computation, suggesting that in diagnosing dynamic system (in which traditional dependency- based diagnosis techniques such as GDE (de Kleer & Williams, 1987) have computational problems), it may be sufficient to diagnose the system based on a snapshot of its current state (as observed), without attempting to track the dependencies in state changes with time. This makes the computation much more efficient, without sacrificing the quality of the diagnosis in many cases. We can view our use of the teamwork model, as using such a set of intra- state constraints which can be verified based on current observations to detect failure (though the diagnosis may require us to keep track of the last state in which the agents were in agreement). However, our work on Eavesdropper goes a step further by using these intra-state

constraints to focus the hypotheses being maintained about the system over time (i.e., over inter-state constraints).

Schroeder and Wagner (1997) have proposed a technique for distributed diagnosis by cooperating agents who receive requests for tests and diagnoses, and send responses to other agents. They each construct a global diagnosis based on the local ones they produce and receive—with the assumption that no conflicts will occur. Frohlich and Nejdl (1996) investigates a related scheme in which multiple diagnosis agents cooperate via a blackboard architecture in diagnosing a physical system. The agents may use different diagnosis models or systems, but a centralized conflict-resolution agent is employed to handle any conflicts in diagnoses found. Both these approaches do not consider communication costs, uncertainties in the construction of the global view, or communication failures. In contrast, our work uses plan recognition and minimizes the use communications. In addition, while our work does not address conflicts in diagnosis per-se, we have presented a set of conditions which make conflicts irrelevant for detection. For instance, under the conditions specified in Theorem 4, distributed monitoring is guaranteed to provide complete and sound detection, regardless of whether the correct hypotheses were selected by the agents in question.

11.6 Other Related Work

There are a few social measures related to the ATA. Goldberg and Mataric (1997) investigate a multi-robot foraging task and measure *interference*—the amount of time robots spend avoiding each other during task execution. Balch (1998) uses social entropy (Bailey, 1990) to measure *behavioral diversity* in multi-agent tasks of soccer, foraging, and formation-maintenance. Both investigations focus on characterizing the relationships between homogeneity and performance in multi-agent systems within the contexts of specific tasks. In contrast, the ATA value measures teamwork, regardless of heterogeneity of the agents. However, unlike investigations of interference and social entropy, correlation between our proposed ATA measure and task performance remains to be investigated.

Matsubara et al. (Matsubara, Frank, Tanaka-Ishii, Noda, Nakashima, & Hasida, 1998) discuss a RoboCup soccer commentator system which uses a mixture of statistical and graphics-based methods to analyze games in real-time and make natural language comments in a manner similar to human commentators. Their focus is on real-time natural- language generation, not on monitoring, but they make use of soccer-specific measures such as statistics of which player passed to which players and average positions of players on the field, etc. Tanaka et al. (Tanaka, Frank, Noda, & Matsubara, 1999) extend the statistical analysis methods further, by introducing and statistically analyzing 32 different features related to soccer performance. Both of these efforts thus focus on soccer-specific measures of performance, while our work focuses on social measures which are task- and domain-independent.

The idea of using relations between processes to drive diagnosis has been introduced independently in the area of software engineering and debugging. Abramson et al. (Abramson, Foster, Michalakes, & Sosic, 1996) describe the use of Guard, a relative debugger used to help in debugging large scientific applications. The idea underlying Guard is similar to socially-attentive monitoring, in that Guard allows the designer to specify a set of relative assertions-basically certain features or values that should be the same between any two versions of a program and the times at which they should be identical. Guard than runs the two versions and compares their state using the assertions. If any differences are found, the debugger helps the user isolate the problem. Unlike relative debugging though, socially-attentive monitoring has the advantage of having many theories that have been explored in the literature that can provide the relationship data a-priori, without knowing about the actual task of the agents. Socially-attentive monitoring is also not limited to simple comparisons, and allows for more expressive and sophisticated relationship checks to be used. Our investigation of socially-attentive methods have addressed uncertainty in monitoring, leading to the monitoring selectivity problem, which is not raised in the relative debugging tasks. We provide analytical guarantees on the effectiveness of our methods, which are not available for the relative debugging task.

O'Leary (1994) provides a survey and exploration of the relation between decision-making correctness and the consensus (majority) decision-making scheme in a group of agents. Given the distribution across the agents of probability of being

competent, and the probability of being correct, O'Leary shows a set of implications and conditions for the applicability of the consensus decision making scheme in a group of agents. The agents are assumed to be completely independent of each other, and the decision is made in a single round of voting. It would therefore seem like no relationship is used by the agents, however this is not the case. A socially-attentive monitoring approach could be used to verify that the decision arrived at by the group (regardless of what it is and whether it is correct) is arrived at by consensus, and not by some other means.

Work on human teamwork assessment and evaluation (e.g., Burns, Salas, & Cannon-Bowers, 1993; Volpe et al., 1996; Zalesny, Salas, & Prince, 1995) lends credibility to an underlying hypothesis of socially-attentive monitoring, that relationship models can be independent of the task. In particular, this trend of research has differentiated between task-work and teamwork in attempting to assess human teams' performance and characteristics, and investigated of training human-teams in multiple tasks to improve teamwork.

The teamwork and role-similarity comparison test was somewhat inspired by Social Comparison Theory (Festinger, 1954) whose first three axioms are as follows (quoted in (Newell, 1990, p. 497)):

1. Every agent has a drive to evaluate its opinions and abilities.

2. If the agent can't evaluate its opinions and abilities objectively, then it compares them against the opinions and abilities of others.

3. Comparing against others decreases as the difference with others increases.

The numerous failure reports we have collected empirically demonstrate the very real need of agents in dynamic, unpredictable domains to evaluate themselves by monitoring their execution (first axiom). Current approaches emphasizing the designer as a source of information against which to compare the agent's performance fit naturally under the title of objective sources for the agent's self-evaluation. Our teamwork and role-similarity relationship test is inspired by the remaining parts of the axioms–allowing the agent to compare its own abilities and opinions (i.e., behavior, beliefs, and goals) to those of others, and considering the weight put on these comparisons (third axiom).

Work on Social Comparison Theory has continued since the original work by Festinger (1954), and can be used to explain many issues in human group behavior. For example, groups tend to adopt the majority views when a clear solution is not available, but adopt such a solution despite a majority if at least a minority of agents knows of it (Baron & Byrne, 1997, p. 456). In socially-attentive monitoring terms, it would seem that the importance and use of socially-attentive monitoring rises as other measures become unavailable. This agrees with Festinger's second axiom.

11.7 Summary

Due to the centrality of the monitoring selectivity problem in multi-agent monitoring, and due to the broad range of capabilities required by the systems carrying out multi-agent monitoring tasks, it is difficult to account for all literature reporting on related investigations. Nevertheless, the previous sections demonstrate that the algorithms and analytical proofs we provide in earlier chapters build effectively on existing work, complementing it and adding to it. In particular, our exploration of teamwork failure-detection algorithms, multi-agent monitoring based on plan recognition, and diagnosis all differ significantly from existing investigations.

Chapter 12

Conclusions and Future Work

The work presented in this dissertation is motivated by practical concerns. The monitoring selectivity problem is a key problem in practical deployment of multi-agent monitoring systems: The practical infeasibility of continuous, complete monitoring of multiple agents, leads to uncertainty in monitoring; This uncertainty, in turn, leads to degraded monitoring performance, both in efficiency and in accuracy. While the need to monitor one's teammates has been recognized repeatedly in the past (Jennings, 1993; Grosz & Kraus, 1996; Tambe, 1997), the monitoring selectivity problem remained largely unaddressed (Jennings, 1993; Grosz & Kraus, 1996).

We have begun our investigation of the monitoring selectivity problem in the first task, relationship failure detection, as a result of our observation that failures continue to occur despite our agents' use of monitoring conditions and communications. Analysis of the failures revealed that agents were not sufficiently informed about each other's state, and yet, practical concerns prevented continuous communications-based monitoring as a solution to this problem. In the second monitoring task, the distribution of the monitored team a-priori limits the observations available to the monitor, making the monitoring selectivity problem even more challenging, especially given the higher accuracy requirements. Moreover, an increasing number of agents being monitored poses challenges to the computational requirements of the monitoring algorithms.

We provide key answers to the monitoring selectivity problem, relying on socially-attentive monitoring methods that exploit knowledge of the relationships that hold between agents, and the procedures that maintain them. Within the context of monitoring in teams, we demonstrate that teamwork failures can be detected effectively

even with uncertain, limited, knowledge of team-members' states. In exploring off-line monitoring in TEAMORE, we have shown that monitoring other agents can be traded for monitoring the environment. We furthermore demonstrate that effective general monitoring (state identification and tracking) can be achieved despite reliance on limited observations of monitored agents and limited computational resources.

Our focus on monitoring agreements on joint plans stems from the centrality of similar notions of agreement in agent and human teamwork literature (Jennings, 1995; Grosz & Kraus, 1996; Volpe et al., 1996; Tambe, 1997). We attempted to demonstrate how the results and techniques can be applied in other domains, and explicitly pointed out necessary conditions and assumptions which underly theorems and algorithms. This allows an agent designer to verify the suitability of the techniques to specific target application domains.

12.1 Contributions

In Part I, we explore a family of teamwork failure-detection algorithms, providing analytical soundness and completeness guarantees, which were verified empirically in the ModSAF and RoboCup simulation domain. Both ModSAF and RoboCup are dynamic, complex, multi-agent domains that involve many uncertainties in perception and action:

- We show analytically that centralized active teamwork monitoring provides failure-detection that is either complete and unsound, or sound and incomplete. However, centralized teamwork monitoring requires multiple hypotheses and monitoring of all team-members.

- We show that in contrast, distributed teamwork monitoring can exploit the local state of agents, which is not available to the centralized algorithm, to provide complete and sound failure-detection, despite using a simpler algorithm and monitoring only key agents in a team.

- We provide initial diagnosis procedures, and initial results in monitoring mutual-exclusion and role-similarity relationships.

- We present a general framework for socially-attentive monitoring, and implement it to empirically validate the analytical results in the ModSAF domain. We further demonstrate the generality of the framework by applying it in the RoboCup domain, in service of off-line quantitative analysis.

In Part II, we empirically explored a number of monitoring methods which provide increased accuracy in monitoring a distributed team, while using constrained resources. The evacuation rehearsal domain, used in this exploration, integrates heterogeneous agents from different groups, and offers very limited observations which can be utilized for monitoring:

- We empirically demonstrate that knowledge of the relationships that hold in a monitored team can lead to significant boost in monitoring accuracy via the minimum-number-of-failure heuristic (e.g., coherence).

- We demonstrate that knowledge of the procedures used by the team to maintain the relationships can be used to significantly boost accuracy further. Overall, the techniques we present increased monitoring accuracy from less than 4% to 72-97%.

- We explore a number of monitoring algorithms which present a trade-off between expressivity (in being able to explicitly model failure hypotheses) and efficiency. In particular, we present the YOYO* algorithm which sacrifices some expressivity to monitor the parallel activities on an entire team in a structure which remains constant in size despite any increase in the number of agents monitored.

12.2 Future Work

We made several references to additional areas in which we would like to conduct further investigations. These include further investigation and formalization of socially-attentive diagnosis procedures, the use of learning to automatically acquire the relationship knowledge used in the techniques described, and relaxing the assumptions underlying the techniques. We discuss those areas in more detail below.

We have shown in Chapter 4 that the diagnosis procedure presented in Section 3.4 is sensitive to the accuracy of the monitoring results. Unfortunately, such sensitivity discourages practical applications of the algorithm as it currently stands. One way to tackle this problem is to improve the accuracy of the monitoring. In particular, if we assume a distributed monitoring configuration, then the distribution and knowledge about what agents detected a failure can be used to constrain the possible hypotheses and improve accuracy. The techniques discussed in Part II may be useful as well. Another approach would be to tackle the diagnosis procedures themselves. It may be that certain failures can be identified and diagnosed despite uncertainty in monitoring.

To this aim, we are currently pursuing an attempt to formalize the diagnosis problem in multi-agent settings. We hope that formalization of the diagnosis problem will lead towards better understanding of the unique solutions that may be possible in multi-agent settings, as well as allow us to bring to bear familiar techniques from the diagnosis field.

A topic which we plan to investigate in depth is the strong requirements of the distributed teamwork monitoring algorithm in terms of observability. In order to provide its soundness and completeness guarantees, the distributed algorithm relies on the ability of all team-members to monitor the key agents. We are investigating ways to relax this requirement while still providing guaranteed results. Certainly, the results we provide in Part II alleviate this constraint somewhat, in that they allow some temporal uncertainty to exist, such that the requirements can be relaxed to allow for some gaps in the observability of key agents. However, a more principled approach is needed to relax this requirement further.

We hope to address this problem by looking at two possible techniques: (a) *monitoring chains*, in which one agent is monitoring another, which in turn is monitoring a key agent. Such chains may be useful in detecting failures despite non-observability of some agents; and (b) *monitoring timing*, in which the agents are not observed continuously, but only at certain points in time. Monitoring timing techniques have been explored in the past in monitoring a single agent (Hansen & Zilberstein, 1996), and in monitoring the environment (Cohen, Atkin, & Hansen, 1994; Atkin & Cohen, 1996). We intend to investigate their use in multi-agent settings.

We have demonstrated the usefulness of using learning to automatically acquire the knowledge required for socially attentive monitoring methods. In particular, we have shown that the communication responses made by the team can be learned. We intend to continue our investigation of this approach, to improve the learned predictions, and to attempt learning of other knowledge. Early experiments show that it may be possible to learn the team-hierarchy, and parts of the plan hierarchy as well.

Two key assumptions which we have made in this work are high on our list of investigation. The first assumption involved assuming no failures in monitored actions and in our observation, i.e., that if an agent is observed to have taken an action, than this action was intended and executed by the monitored agent. For example, if a helicopter is observed as landing, we have assumed that indeed the landing action was intended and carried out, and that our sensing of the action was correct. Relaxing this assumption is important to further demonstrate the practicality of our approach.

A second assumption that we have made is that the plan-library used for plan recognition is complete and correct, i.e., that any plan executed by the monitored agents is in the library use for recognition. Although this is a common assumption in plan recognition literature (e.g., Kautz & Allen, 1986), we feel that it is inherently limiting for multi-agent monitoring purposes. In general, one cannot expect to have a complete and correct plan-library of monitored agents. For instance, when monitoring a RoboCup soccer game, one cannot expect to know the plan-library of opponent teams. We are considering several ways to approach this problem in the context of specific monitoring tasks. The idea would be to guarantee certain monitoring performance despite the use of less-than-perfect plan-libraries.

12.3 Acknowledgment

This research was generously supported in part by NSF Grant ISI-9711665, and in part by AFOSR contract #F49620-97-1-0501.

Bibliography

Abramson, D., Foster, I., Michalakes, J., & Sosic, R. (1996). Relative debugging: A new methodology for debugging scientific applications. *Communications of the ACM*, *39*(11), 69–77.

Ambros-Ingerson, J. A., & Steel, S. (1988). Intergrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)* Minneapolis/St. Paul, MN. AAAI Press.

Ando, T. (1998). Refinement of soccer agents' positions using reinforcement learning. In Kitano, H. (Ed.), *RoboCup-97: Robot soccer world cup I*, Vol. 1395 of *LNAI*, pp. 373–388. Springer-verlag.

Atkin, M. S., & Cohen, P. R. (1996). Monitoring strategies for embedded agents: Experiments and analysis. *Journal of Adaptive Behavior*, *4*(2), 125–172.

Atkins, E. M., Durfee, E. H., & Shin, K. G. (1997). Detecting and reacting to unplanned-for world states. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 571–576 Providence, RI. AAAI Press.

Bailey, K. D. (1990). *Social Entropy Theory.* State University of New York Press.

Balch, T. (1998). *Behavioral Diversity in Learning Robot Teams.* Ph.D. thesis, Georgia Institute of Technology.

Baron, R. A., & Byrne, D. (1997). *Social Psychology.* Allyn & Bacon.

Burns, J. J., Salas, E., & Cannon-Bowers, J. A. (1993). Team training, mental models, and the team model trainer. In *Advancements in Integrated Delivery Technologies* Denver, CO.

Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. F., & Ceranowicz, A. Z. (1993). Modsaf behavior simulation and control. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Reresentation* Orlando, Florida. Institute for Simulation and Training, University of Central Florida.

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

Cohen, P. R., Amant, R. S., & Hart, D. M. (1992). Early warnings of plan failure, false positives, and envelopes: Experiments and a model. Tech. rep. CMPSCI Technical Report 92-20, University of Massachusetts.

Cohen, P. R., Atkin, M. S., & Hansen, E. A. (1994). The interval reduction strategy for monitoring cupcake problems. In Cliff, D., Husbands, P., Meyer, J.-A., & Wilson, S. W. (Eds.), *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*.

Cohen, P. R., & Levesque, H. J. (1991). Teamwork. *Nous*, *35*.

de Kleer, J., & Williams, B. C. (1987). Diagnosing multiple faults. *Artificial Intelligence*, *32*, 97–130.

Devaney, M., & Ram, A. (1998). Needles in a haystack: Plan recognition in large spatial domains involving multiple agents. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 942–947 Madison, WI.

Doyle, R. J. (1995). Determining the loci of anomalies using minimal causal models. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 1821–1827 Montreal, Quebec, Canada.

Doyle, R. J., Atkinson, D. J., & Doshi, R. S. (1986). Generating perception requests and expectations to verify the execution of plans. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*.

Doyle, R. J., Chien, S. A., Fayyad, U. M., & Wyatt, E. J. (1993). Focused real-time systems monitoring based on multiple anomaly models. In *International Qualitative Reasoning Conference (QR'93)* Eastsound, WA.

Dressler, O. (1994). Model-based diagnosis on board: Magellan-mr inside. In *the International Workshop on Principles of Diagnosis (DX-94)*.

Durfee, E. H. (1995). Blissful ignorance: Knowing just enough to coordinate well. In *Proceedings of the First International Conference on Multiagent Systems (ICMAS-95)*, pp. 406–413.

Fenster, M., Kraus, S., & Rosenschein, J. S. (1995). Coordination without communication: Experimental validation of focal point techniques. In *Proceedings of the First International Conference on Multiagent Systems (ICMAS-95)*, pp. 102–108 California, USA.

Festinger, L. (1954). A theory of social comparison processes. *Human Relations*, *7*, 117–140.

Firby, R. J. (1987). An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*.

Frohlich, P., & Nejdl, W. (1996). Resolving conflicts in distributed diagnosis. In Wahlster, W. (Ed.), *the 12th Europeach Conference on Artificial Intelligence (ECAI-96)*. John Wiley & Sons, Inc.

Goldberg, D., & Mataric, M. J. (1997). Interference as a tool for designing and evaluating multi-robot controllers. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 637–642 Providence, RI. AAAI Press.

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

Grosz, B. J., & Kraus, S. (1999). The evolution of SharedPlans. In Wooldridge, M., & Rao, A. (Eds.), *Foundations and Theories of Rational Agency*, pp. 227–262.

Grosz, B. J., & Sidner, C. L. (1990). Plans for discourse. In Cohen, P. R., Morgan, J., & Pollack, M. (Eds.), *Intentions in Communication*, pp. 417–445. MIT Press, Cambridge, MA.

Halpern, J. Y., & Moses, Y. (1990). Knowledge and common knowledge in a distributed environment. *distributed computing*, *37*(3), 549–587.

Hamscher, W., Console, L., & de Kleer, J. (Eds.). (1992). *Readings in Model-Based Diagnosis*. Morgan Kaufmann Publishers, San Mateo, CA.

Hansen, E. A., & Zilberstein, S. (1996). Monitoring the progress of anytime problem solving. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pp. 1229–1234. AAAI Press.

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

Huber, M. J., & Durfee, E. H. (1995). On acting together: Without communication. In *Working Notes of the AAAI Spring Symposium on Representing Mental States and Mechanisms*, pp. 60–71 Stanford, CA.

Huber, M. J., & Hadley, T. (1997). Multiple roles, multiple teams, dynamic environment: Autonomous netrek agents. In Johnson, W. L. (Ed.), *Proceedings of the First International Conference on Autonomous Agents (Agents-97)*, pp. 332–339 Marina del Rey, CA. ACM Press.

Hudlicka, E., & Lesser, V. R. (1987). Modeling and diagnosing problem-solving system behavior. *IEEE Transactions on System, Man, and Cybernetics*, *17*(3), 407–419.

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

Intille, S. S., & Bobick, A. F. (1999). A framework for recognizing multi-agent action from visual evidence. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 518–525. AAAI Press.

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

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

Kautz, H. A., & Allen, J. F. (1986). Generalized plan recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pp. 32–37. AAAI press.

Kinny, D., Ljungberg, M., Rao, A., Sonenberg, E., Tidhar, G., & Werner, E. (1992). Planned team activity. In Castelfranchi, C., & Werner, E. (Eds.), *Artificial Social Systems, Lecture notes in AI 830*, pp. 227–256. Springer Verlag, New York.

Kitano, H., Tambe, M., Stone, P., Veloso, M., Coradeschi, S., Osawa, E., Matsubara, H., Noda, I., & Asada, M. (1997). The RoboCup synthetic agent challenge '97. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-97)* Nagoya, Japan.

Klein, M., & Dellarocas, C. (1999). Exception handling in agent systems. In *Proceedings of the Third International Conference on Autonomous Agents (Agents-99)*. ACM Press.

Kraus, S., Sycara, K., & Evenchik, A. (1998). Reaching agreements through negotiations: a logical model and implementation. *artificial intelligence*, *104*(1-2), 1–69.

Kuniyoshi, Y., Rougeaux, S., Ishii, M., Kita, N., Sakane, S., & Kakikura, M. (1994). Cooperation by observation—the framework and the basic task patterns. In *the IEEE International Conference on Robotics and Automation*, pp. 767–773 San-Diego, CA. IEEE Computer Society Press.

Lesh, N., Rich, C., & Sidner, C. L. (1999). Using plan recognition in human-computer collaboration. In *Proceedings of the Seventh International Conference on User Modelling (UM-99)* Banff, Canada.

Levesque, H. J., Cohen, P. R., & Nunes, J. H. T. (1990). On acting together. In *Proceedings of the Eigth National Conference on Artificial Intelligence (AAAI-90)* Menlo-Park, CA. AAAI Press.

Malone, T. W., & Crowston, K. (1991). Toward an interdisciplinary theory of coordination. Tech. rep. CCS TR#120 SS WP# 3294-91-MSA, Massachusetts Institute of Technology.

Marsella, S. C., Adibi, J., Al-Onaizan, Y., Kaminka, G. A., Muslea, I., Tallis, M., & Tambe, M. (1999). On being a teammate: Experiences acquired in the design of robocup teams.. In *Proceedings of the Third International Conference on Autonomous Agents (Agents-99)*, pp. 221–227 Seattle, WA. ACM Press.

Matsubara, H., Frank, I., Tanaka-Ishii, k., Noda, I., Nakashima, H., & Hasida, K. (1998). Automatic soccer commentary and robocup. In Asada, M. (Ed.), *the Second RoboCup Workshop (RoboCup-98)*, pp. 7–22 Paris, France.

Ndumu, D. T., Nwana, H. S., Lee, L. C., & Collis, J. C. (1999). Visualizing and debugging distributed multi-agent systems. In *Proceedings of the Third International Conference on Autonomous Agents (Agents-99)*. ACM Press.

Newell, A. (1990). *Unified Theories of Cognition.* Harvard University Press, Cambridge, Massachusetts.

O'Leary, D. E. (1994). Models of consensus for multiple agent systems. In *the Tenth Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pp. 447–453 Seattle, WA. Morgan-Kaufmann Publishers.

Parker, L. E. (1993). Designing control laws for cooperative agent teams. In *the Proceedings of the IEEE Robotics and Automation Conference*, pp. 582–587 Atlanta, GA.

Parker, L. E. (1998). ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, *14*(2), 220–240.

Pechoucek, M., Marik, V., & Stepankova, O. (2000). Role of acquaintance models in an agent-based production planning. In Klusch, M., & Kerschberg, L. (Eds.),

Cooperative Information Agents IV, Proceedings of the Fourth International Workshop (CIA-2000), No. 1860 in LNAI, pp. 179–190. Springer Verlag.

Pynadath, D. V., Tambe, M., Chauvat, N., & 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")*, pp. 77–91.

Rao, A. S. (1994). Means-end plan recognition – towards a theory of reactive recognition. In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR-94)*, pp. 497–508.

Reece, G. A., & Tate, A. (1994). Synthesizing protection monitors from causal structure. In *Proceedings of Artificial Intelligence Planning Systems (AIPS-94)* Chicago, IL.

Rich, C., & Sidner, C. L. (1997). COLLAGEN: When agents collaborate with people. In Johnson, W. L. (Ed.), *Proceedings of the First International Conference on Autonomous Agents (Agents-97)*, pp. 284–291 Marina del Rey, CA. ACM Press.

Rickel, J., & Johnson, W. L. (1999a). Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *Applied Artificial Intelligence*, *13*, 343–382.

Rickel, J., & Johnson, W. L. (1999b). Virtual humans for team training in virtual reality. In *Proceedings of the Ninth International Conference on Artificial Intelligence in Education*, pp. 578–585. IOS Press.

Schroeder, M., & Wagner, G. (1997). Distributed diagnosis by vivid agents. In *Proceedings of the First International Conference on Autonomous Agents (Agents-97)*, pp. 268–275 Marina del Rey, CA. ACM Press.

Sugawara, T., & Lesser, V. R. (1998). Learning to improve coordinated actions in cooperative distributed problem-solving environments. *Machine Learning*, *33*(2/3), 129–153.

Tambe, M. (1996). Tracking dynamic team activity. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*.

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

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

Tambe, M., & Zhang, W. (1998). Towards flexible teamwork in peristent teams. In *Proceedings of the Third International Conference on Multiagent Systems (ICMAS-98)*.

Tanaka, K., Frank, I., Noda, I., & Matsubara, H. (1999). A statistical perspective on the robocup simulator league: Progress and prospects. In Veloso, M. (Ed.), *the Third RoboCup Workshop (RoboCup-99)*, pp. 217–222 Stockholm, Sweden.

Toyama, K., & Hager, G. D. (1997). If at first you don't succeed.... In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 3–9 Providence, RI.

Van Beek, P., & Cohen, R. (1991). Resolving plan ambiguity for cooperative response generation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 938–944 Sydney, Australia.

Veloso, M., Pollack, M. E., & Cox, M. T. (1998). Rationale-based monitoring for planning in dynamic environments. In *Proceedings of Artificial Intelligence Planning Systems (AIPS-98)* Pittsburgh, PA.

Volpe, C. E., Cannon-Bowers, J. A., & Salas, E. (1996). The impact of cross-training on team functioning: An empirical investigation. *human factors*, *38*(1), 87–100.

Washington, R. (1998). Markov tracking for agent coordination. In *Proceedings of the Second International Conference on Autonomous Agents (Agents-98)*, pp. 70–77 Minneapolis/St. Paul, MN. ACM Press.

Wilkins, D. E. (1988). *Practical planning: extending the classical AI Planning paradigm.* the morgan-kaufmann series in representation and reasoning, ed. R. J. Brachman. morgan-kaufmann publishers, Inc.

Zalesny, M. D., Salas, E., & Prince, C. (1995). Conceptual and measurement issues in coordination: Implications for team behavior and performance. *Research in Personnel and Human Resources Management*, *13*, 81–115.

Appendix A

List of Selected Publications

- Kaminka, G. A., and Tambe, M. 2000. Robust Multi-Agent Teams via Socially-Attentive Monitoring. In *Journal of Artificial Intelligence Research (JAIR)*. Volume 12, pp. 105-147.
- Marsella, C. S.; Adibi J.; Al-Onaizan, Y.; Kaminka G. A.; Muslea, I; Tallis, M.; and Tambe, M. On Being a Teammate: Experiences Acquired in the Design of RoboCup Teams. In the *Journal of Autonomous Agents and Multi-Agent Systems*. To appear in the "Best of Agents'99" Special Issue.
- Tambe, M.; Pynadath, D. V.; Chauvat, N.; Das, A.; and Kaminka, G. A. 2000. Adaptive Agent Integration Architectures for Heterogeneous team-members. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-2000)*. To appear.
- Kaminka, G. A., and Tambe, M. 1999. I'm OK, You're OK, We're OK: Experiments in Distributed and Centralized Socially Attentive Monitoring. In *Proceedings of the Third International Conference on Autonomous Agents (Agents-99)*, Seattle, Washington.
- Tambe, M.; Adibi, J.; Al-Onaizan, Y; Erdem, A.; Kaminka, G. A.; Marsella C. S.; and Muslea, I. 1999. Building Agent Teams Using an Explicit Teamwork Model and Learning. *Artificial Intelligence*. Volume 111, pp. 215-239.
- Marsella, C. S.; Adibi J.; Al-Onaizan, Y.; Kaminka G. A.; Muslea, I; Tallis, M.; and Tambe, M. 1999. On Being a Teammate: Experiences Acquired in the Design of RoboCup Teams. In Proceedings of *the Third International Conference on Autonomous Agents (Agents-99)*, Seattle, Washington.
- Tambe, M.; Kaminka, G. A.; Marsella, C. S.; Muslea, I.; and Raines, T. 1999. Two fielded teams and two experts: A RoboCup Challenge Response from the Trenches, In *Proceedings of the International Conference on Artificial Intelligence (IJCAI-99)*. pp. 276-281. July 31-August 6, Stockholm, Sweden.

- Kaminka, G. A., and Tambe, M. 1998 What's Wrong With Us? Improving Robustness through Social Diagnosis. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)* Madison, Wisconsin, July 26-30, 1998. An earlier version appears in *Proceedings of the Agents-98 Workshop on Agents in Interaction — Acquiring Competence Through Imitation*, Minneapolis/St. Paul, May 9-13, 1998.

- Kaminka, G. A.; Tambe, M., and Hopper, C. M. 1998 The Role of Agent-Modeling in Agent Robustness. In *AI Meets the Real-World: Lessons Learned (AIMTRW-98)*, Stamford, CT, September 1998.

- Kaminka, G. A., and Tambe, M. 1998 Social Comparison for Failure Detection and Recovery. In *Intelligent Agents IV: Agents, Theories, Architectures and Languages (ATAL), Proceedings of the 4th International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, Lecture Notes in Artificial Intelligence 1365, Springer-Verlag, pp. 127-141

Appendix B

Proofs

(# 2, page 34). Let a monitoring agent A monitor a simple team T. If A's team-modeling of T is complete, and A uses a maximally team-coherent hypothesis for detection, then the teamwork failure detection results are sound.

Proof. We will show that any failure that occurs is detected, and thus that all failures will be detected. Let $a_1, \ldots, a_n$ be the agent members of T. Each agent a_i is executing some plan P_i $(1 \leq i \leq n)$. Thus collectively, the group is executing $(P_1, \ldots, P_n)$. If a failure has occurred, then there are two agents $a_k, a_j, 1 \leq j, k \leq n$ such that a_j is executing plan P_j and a_k is executing plan P_k and $P_j \neq P_k$. Since A's team-modeling is complete, the correct hypothesis $(P_1, \ldots, P_j, \ldots, P_k, \ldots P_n)$ will in the set of team-modeling hypotheses. Since A will choose a maximally team-incoherent hypothesis, either it will choose the correct hypothesis, which is more incoherent than a hypothesis implying no failure has occurred, or that it will select a hypothesis with greater incoherence hypothesis (or equivalent level). In any case, a failure would be detected, and the detection procedure is complete. □

(# 1, page 40). Suppose a simple team T is self-monitoring (all members of the team monitor each other) using the maximally team-coherent heuristic (and under the assumption that for each agent, team-modeling is complete). A monitoring agent A_1 who is a member of T and is executing P_1 would detect a failure in maintaining teamwork relationships with an agent A_2 (also a member of T) executing a different plan P_2, if A_2 has an observably different role in P_1 and P_2.

Proof. A_1 knows that it is executing P_1. Since all members of T monitor each other and themselves, A_1 is monitoring A_2, who has an observably different role in P_1 and P_2. Since A_2 is executing P_2, and following the observably different role, $P_1 \notin M(A_1, A_2/P_2)$. Therefore from the perspective of A_1, it cannot be the case that it assigns P_1 in any *agent-modeling* hypothesis, and therefore any *team-modeling* hypothesis that A_1 has will have A_1 executing P_1, and A_2 executing some plan other than P_1. In other words, from A_1's perspective there is no team-coherent hypothesis, and so a difference would be detected between A_1 and A_2. □

(# 5, page 53). Let a monitoring agent A monitor mutual-exclusion relationships in a group of agents G. If A's modeling of G is complete, and A uses a maximally team-*in*coherent hypothesis for detection, then the failure detection results are sound.

Proof. We will show that if no failure has occurred, none will be detected, and thus that any failure that is detected is in fact a failure. Let $a_1, \ldots, a_n$ be the agent members of G. Each agent a_i is executing some plan P_i $(1 \leq i \leq n)$. Thus collectively, the group is executing $(P_1, \ldots, P_n)$. If no failure has occurred, then each agent is executing a different plan $(i \neq j \Rightarrow P_i \neq P_j)$. Since A's group-modeling is complete, the correct hypothesis is going to be in the set of group-modeling hypotheses H. Since it is a maximally incoherent hypothesis, either it will be selected, or that a different hypothesis *of the same coherence level* will be selected. Any hypothesis with the same coherence level as the correct one implies no failure is detected. Thus the detection procedure is sound. □

(# 6, page 53). Let a monitoring agent A monitor mutual-exclusion relationships in a group of agents G. If A's modeling of G is complete, and A uses a maximally team-coherent hypothesis for detection, then the failure detection results are complete.

Proof. We will show that any failure that occurs is detected, and thus that the procedure is complete. Let $a_1, \ldots, a_n$ be the agent members of G. Each agent a_i is executing some plan P_i $(1 \leq i \leq n)$. Thus collectively, the group is executing $(P_1, \ldots, P_n)$. If a failure has occurred, then there are two agents $a_k, a_j, 1 \leq j, k \leq n$ such that a_j is executing plan P_j and a_k is executing plan P_k and $P_j = P_k$. Since A's group-modeling is complete, the correct hypothesis $(P_1, \ldots, P_j, \ldots, P_k, \ldots P_n)$ will in the set of group-modeling hypotheses. Since A will choose a maximally team-coherent hypothesis, either it will choose the correct hypothesis, which is more coherent than a hypothesis implying no failure has occurred, or that it will select a hypothesis with greater coherence hypothesis (or equivalent level). In any case, a failure would be detected. Therefore, the detection procedure is complete. □

(# 7, page 67). Suppose we have k agents, each with its associated plan-horizon H_i, where $1 \leq i \leq k$. The number of coherent team-state hypotheses when combining H_i is no greater than $\min\{size(H_i)\}$, where $size(H_i)$ is the number of hypotheses in the i'th agent plan-horizon H_i.

Proof. Let H_{min} be a plan-horizon with $size(H_{min}) = \min\{size(H_i)\}$. Let P be a team-plan that is in some H_i but not in H_{min} (i.e., $H_i \neq H_{min}$). There are two cases:
case (i): If no such plan P exists, then all plan-horizons H_i are equal, and the number of coherent hypotheses cannot be more than the size of H_{min}.
case (ii): If such a plan P exists, then it cannot be a coherent hypothesis, since it is not shared by H_{min} and H_i.

Therefore any hypothesized plan not in H_{min} is necessarily not coherent, and therefore that the number of coherent hypotheses cannot be greater than the number of hypotheses in H_{min}, which is $size(H_{min})$. □

Appendix C

Efficient Single-Agent Probabilistic plan recognition

This appendix presents a mechanism, co-developed with David V. Pynadath, that uses only the messages sent by a single team member for recognizing its plans, based on predictions of the duration plan-execution. For instance, if we observe a message about the initiation of `Fly-Flight-Plan`, then we know from Figure 8.1b 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 C.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 C.2.

C.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 $1/(\text{average duration of } X)$. Given this model of plan duration, the probability of the plan's completion between times t and $t+1$ is simply $\Pr(done(X,t)|X_t) = 1 - e^{-\lambda_X}$.

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:

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

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

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 child. In Figure 8.1, 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 (i.e., multiple sub-plans that are to be executed by different sub-teams in parallel), we compute the probability over the 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 C.1 presents the pseudo-code for the overall propagation computations, calling the PROPAGATE-DOWN function for this downward update.

C.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

Algorithm C.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$(Y, \rho\, b, M)$

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 C.1, except that we are now conditioning on observing a message. Algorithm C.2 presents the pseudo-code for the complete procedure for incorporating observational evidence.

Algorithm C.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$(X, b'(X, \neg block), b, M)$

C.3 Individual Agent Recognition Complexity

The pseudo-code of Algorithms C.2–C.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.

Appendix D

Socially-Attentive Monitoring Algorithms

We bring here the algorithms (in pseudo-code) for the RESL plan recognition algorithm, the comparison test supporting detection in both simple and non-simple teams, and the monitoring algorithms for the centralized and distributed cases.

D.1 RESL

RESL works by first expanding the complete operator hierarchy for the agents being modeled, tagging all plans as non-matching. All plans' preconditions and termination conditions are flagged as non-matching as well. All plans' actions are set to be used as expectations on behavior. After initializing the plan-recognition hierarchy for each monitored agent, observations of an agent are continuously matched against the actions expected by the plans. Plans whose expectations match observations are tagged as matching, and these flags are propagated along the hierarchy, up and down, so that complete paths through the hierarchy are flagged as matching or not. These paths specify the possible matching interpretations of the observations. In addition, precondition and termination conditions are flagged as true or not, signifying the inferred appropriate belief by the modeled agents. This process is described in Algorithm D.1.

D.2 Detection of Failure, Centralized and Distributed Teamwork Monitoring

Algorithm D.2 shows how comparison of hierarchical plans is carried out. We limit ourselves here to simple-teams. The algorithm accepts as input two sets of hierarchical plan hypotheses, $hierarchies_1$, $hierarchies_2$, corresponding to two monitored agents (for clarity, the algorithms assume only two agents. The generalization to n agents is straightforward). The algorithm also accepts a policy flag, `Policy`. An `OPTIMISTIC` policy causes the algorithm to use maximal team-coherence to provide

Algorithm D.1 RECOGNIZE(**agent** A)

```
1: let O be the current observations of A
2: for each plan P with expected observations E do
3:    if O matches E then
4:       flag P as matching
5:    else
6:       flag P as failing to match
7: for each plan that is flagged as matching do
8:    flag its parent plans as matching {propagate matches}
9: for each plan P do
10:   if all children plans of P are flagged as failing to match then
11:      flag P as failing to match {propagate non-matches}
12: return all matching plans
```

sound, but incomplete detection. A `PESSIMISTIC` policy causes the algorithm to use maximal team-incoherence to provide complete, but unsound detection.

Algorithm D.2 DETECT($hierarchies_1, hierarchies_2, policy$)

```
1: depth ← 0 {look for top-most failure}
2: while plans at depth depth of hierarchies_1, hierarchies_2 are team plans do
3:    if policy =OPTIMISTIC then
4:       let plan_1, plan_2 be two maximally coherent plans at level depth of
         hierarchies_1, hierarchies_2
5:    else
6:       let plan_1, plan_2 be two maximally incoherent plans at level depth of
         hierarchies_1, hierarchies_2
7:    if plan_1 ≠ plan_2 then
8:       return FAILURE
9:    else
10:      if if bottom of hierarchies_1, hierarchies_2 reached then
11:         return NO_FAILURE
12:      else
13:         depth ← depth + 1
```

With the aid of Algorithm D.2, we can now define the centralized and distributed failure detection algorithms. The centralized teamwork monitoring algorithm (Algorithm D.3) utilizes Algorithm D.2 twice, checking for failures with both `PESSIMISTIC` and `OPTIMISTIC` policies. If the results of both policies agree, they are certain. If the results do not agree, (i.e., the `PESSIMISTIC` policy causes a failure to be detected, while the `OPTIMISTIC` policy causes no failure to be detected), then the monitoring agent cannot be certain that a failure has taken

place, and therefore needs to verify the failure. Algorithm D.3 therefore returns `FAILURE, NO_FAILURE, POSSIBLE_FAILURE`.

Algorithm D.3 CENTRALIZED-MONITORING

```
1: for every two monitored agents A_1, A_2 do
2:    H_1 ←RECOGNIZE(A_1)
3:    H_2 ←RECOGNIZE(A_2)
4:    OptimisticResult ← DETECT(H_1, H_2, OPTIMISTIC)
5:    PessimisticResult ← DETECT(H_1, H_2, PESSIMISTIC)
6:    if OptimisicResult = PessimisticResult =FAILURE then
7:       return FAILURE
8:    else
9:       if OptimisticResult ≠ PessimisticResult then
10:         return POSSIBLE_FAILURE
```

The distributed monitoring algorithm makes is not given is pseudo-code form, because it is nothing more than a call to Algorithm D.2 with an `OPTIMISTIC` policy parameter. Its power is derived from the fact that all members of the team are using it to monitor the key agents of the team.

D.3 Probabilistic version of YOYO*

A probabilistic version of the YOYO* algorithm is presented below (Algorithm D.4). 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 C.1) and INCORPORATE-EVIDENCE (Algorithm C.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`.

Algorithm D.4 YOYO*(**plan hierarchy** M, **team-hierarchy** H)

```
1: Loop forever:
2: if message m received–new plan S team T,  then
3:    INCORPORATE-EVIDENCE(m, current beliefs, M, T)
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 who is not to be taken by T do
8:          SCALE(the subtree roots at the child), so its state probabilities sum up
            to the new probability of A
9:       tmp ← parent_of(tmp)
10: else
11:    PROPAGATE-FORWARD in M
```

First, the new evidence is incorporated for the transport team. Among other changes, the probability of the plan `Landing-Zone-Maneuvers` 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` go up, based on evidence from the TRANSPORT team. The process then continues to `Execute-Mission`, etc.