MODELING HUMANS AS PEERS AND SUPERVISORS IN COMPUTING SYSTEMS THROUGH RUNTIME MODELS

by

CHRISTOPHER ZHONG

M.S., Kansas State University, 2006

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
2012

Abstract

There is a growing demand for more effective integration of humans and computing systems, specifically in multiagent and multirobot systems. There are two aspects to consider in human integration: (1) the ability to control an arbitrary number of robots (particularly heterogeneous robots) and (2) integrating humans as peers in computing systems instead of being just users or supervisors.

With traditional supervisory control of multirobot systems, the number of robots that a human can manage effectively is between four and six [17]. A limitation of traditional supervisory control is that the human must interact individually with each robot, which limits the upper-bound on the number of robots that a human can control effectively. In this work, I define the concept of "organizational control" together with an autonomous mechanism that can perform task allocation and other low-level housekeeping duties, which significantly reduces the need for the human to interact with individual robots.

Humans are very versatile and robust in the types of tasks they can accomplish. However, failures in computing systems are common and thus redundancies are included to mitigate the chance of failure. When all redundancies have failed, system failure will occur and the computing system will be unable to accomplish its tasks. One way to further reduce the chance of a system failure is to integrate humans as peer "agents" in the computing system. As part of the system, humans can be assigned tasks that would have been impossible to complete due to failures.

MODELING HUMANS AS PEERS AND SUPERVISORS IN COMPUTING SYSTEMS THROUGH RUNTIME MODELS

by

CHRISTOPHER ZHONG

M.S., Kansas State University, 2006

A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
2012

Approved by:

Major Professor
Scott A. DeLoach

Copyright

Christopher Zhong

2012

Abstract

There is a growing demand for more effective integration of humans and computing systems, specifically in multiagent and multirobot systems. There are two aspects to consider in human integration: (1) the ability to control an arbitrary number of robots (particularly heterogeneous robots) and (2) integrating humans as peers in computing systems instead of being just users or supervisors.

With traditional supervisory control of multirobot systems, the number of robots that a human can manage effectively is between four and six [17]. A limitation of traditional supervisory control is that the human must interact individually with each robot, which limits the upper-bound on the number of robots that a human can control effectively. In this work, I define the concept of "organizational control" together with an autonomous mechanism that can perform task allocation and other low-level housekeeping duties, which significantly reduces the need for the human to interact with individual robots.

Humans are very versatile and robust in the types of tasks they can accomplish. However, failures in computing systems are common and thus redundancies are included to mitigate the chance of failure. When all redundancies have failed, system failure will occur and the computing system will be unable to accomplish its tasks. One way to further reduce the chance of a system failure is to integrate humans as peer "agents" in the computing system. As part of the system, humans can be assigned tasks that would have been impossible to complete due to failures.

Table of Contents

Table of Contents **vi**

List of Figures **ix**

List of Tables **xi**

Acknowledgements **xii**

1 Introduction **1**
- 1.1 Open Issues . . . 8
- 1.2 Approach . . . 10
  - 1.2.1 Thesis Statement . . . 11
- 1.3 Contributions . . . 12
- 1.4 Overview . . . 12

2 Motivating Example **14**
- 2.1 Scenario . . . 14
- 2.2 Example Script . . . 15
- 2.3 Highlights . . . 19
- 2.4 Summary . . . 20

3 Background **21**
- 3.1 Organization Model for Adaptive Computational Systems (OMACS) . . . 21
- 3.2 Goal Model for Dynamic Systems (GMoDS) . . . 23
  - 3.2.1 Specification Model . . . 24
  - 3.2.2 Instance Model . . . 26
- 3.3 Organization-based Multiagent System Engineering (O-MaSE) . . . 27
- 3.4 Performance Moderator Functions (PMFs) . . . 30
- 3.5 Human-Robot Interaction (HRI) . . . 33
  - 3.5.1 Human Roles . . . 34
  - 3.5.2 Interaction Schemes . . . 35
  - 3.5.3 Robot Automation . . . 41
- 3.6 Summary . . . 43

4 Chazm Model (CzM) **44**
- 4.1 Limitations of OMACS . . . 44
- 4.2 Chazm Model (CzM) . . . 46
- 4.3 Evaluation Process . . . 53

4.4 Multiple Humans Evaluation . . . 54
4.4.1 Experimental Setup . . . 56
4.4.2 Algorithms . . . 59
4.4.3 Attributes Only Results . . . 61
4.4.4 Attributes and Capabilities Results . . . 66
4.4.5 Time Complexity . . . 71
4.5 Multiple Humans Multiple Robots Evaluation . . . 71
4.5.1 Experimental Setup . . . 74
4.5.2 Algorithms . . . 76
4.5.3 Results . . . 78
4.5.4 Time Complexity . . . 81
4.6 Summary . . . 82

5 Organization Control **83**
5.1 Interactions of Organization Control . . . 86
5.2 Intelligent Autonomous Layer (IAL) . . . 90
5.2.1 Scenario . . . 93
5.2.1.1 Goal Model . . . 95
5.2.1.2 Role Model . . . 97
5.2.1.3 Capability Model . . . 99
5.2.1.4 Agent Model . . . 101
5.2.1.5 Organization-Based Agent Architecture (OBAA) . . . 102
5.2.2 Results . . . 102
5.2.2.1 Robots . . . 103
5.2.2.2 Simulation . . . 104
5.2.3 Organization Control Demonstration . . . 106
5.3 Organization Control . . . 110
5.3.1 Assignment Set Manipulation . . . 110
5.3.1.1 Demonstration . . . 112
5.3.2 Goal Modification . . . 119
5.3.2.1 Extension . . . 120
5.3.2.2 Proof . . . 129
5.3.2.3 Demonstration . . . 134
5.4 Summary . . . 140

6 Related Work **144**
6.1 Task Allocation . . . 144
6.1.1 Bioinspired Approaches . . . 145
6.1.2 Organizational Approaches . . . 148
6.1.2.1 Role-Based Approaches . . . 148
6.1.2.2 Market-Based Approaches . . . 150
6.1.3 Knowledge-Based Approaches . . . 153
6.2 Performance Moderator Functions (PMFs) . . . 158

6.3 Supervisory Control . . . 161
6.4 Summary . . . 168

7 Conclusions **169**
7.1 Current State & Issues . . . 169
7.2 Work Done & Contributions . . . 171
7.3 Limitations . . . 172
7.4 Future Work . . . 174

Bibliography **191**

A Acronyms **192**

B Emergency Response Team **196**
B.1 Survey Group . . . 196
B.2 Hazard Identification Group . . . 200
B.3 Victim Rescue Group . . . 203

C Creating Runtime Models **205**

D Time Complexity Analysis **207**
D.1 Random Algorithm . . . 207
D.2 Round Robin Algorithm . . . 209
D.3 Greedy Algorithm . . . 211
D.4 Attributes-Greedy Algorithm . . . 213
D.5 Attributes-Enhanced Algorithm . . . 215
D.6 Brute Force Algorithm . . . 218

E Goal Modification Logs **221**

List of Figures

1.1 Supervisory Control (Sequencing Style) [17] 4
1.2 Supervisory Control 6

3.1 OMACS Model 22
3.2 O-MaSE Methodology Framework [24] 28
3.3 O-MaSE Meta-Model [24] 29

4.1 CzM Model 47
4.2 Conference Management System (CMS) Goal Model 54
4.3 CMS Role Model 55
4.4 Score Difference Graph 63
4.5 Set Commonality Graph 64
4.6 Review Quality Graph 65
4.7 Score Difference Graph 68
4.8 Set Commonality Graph 69
4.9 Review Quality Graph 70
4.10 Retrieval Goal Model 72
4.11 Retrieval Role Model – OMACS 72
4.12 Modified Roles 73
4.13 Cases Graph 79
4.14 Completion Time Graph 80
4.15 Completion Time Graph (±1 Standard Deviation) 81

5.1 Organization Control 85
5.2 Agent Architecture 91
5.3 Intelligent Autonomous Layer (IAL) 92
5.4 Improvised Explosive Devices 93
5.5 Using iRobot PackBot 94
5.6 Scenario 95
5.7 IED Goal Model 96
5.8 IED Role Model 97
5.9 IED Capability Model 100
5.10 IED Agent Model 101
5.11 IED Detection System 104
5.12 Reassignments Example 105
5.13 Designing Control 107
5.14 Designing Control 108
5.15 Scenario Layout 113

5.16 Initial State . . . 114
5.17 Monitoring Interface – Halfway Point . . . 115
5.18 Monitoring Interface – Waiting . . . 116
5.19 Monitoring Interface – Reassignment . . . 117
5.20 Monitoring Interface – Completion . . . 118
5.21 Source of Parameters . . . 125
5.22 Same Set of Keys . . . 126
5.23 Values are from Triggers . . . 127
5.24 Goal Modification . . . 128
5.25 Propagation of Modifications . . . 130
5.26 Partial Goal Model . . . 135
5.27 Reformatted Log of IAL . . . 136
5.28 Reformatted Log of Robots . . . 138
5.29 Waypoint Modification . . . 139
5.30 Reformatted Log of IAL . . . 141
5.31 Reformatted Log of Leader Robot . . . 142

D.1 Pseudo Code – Random Algorithm . . . 208
D.2 Pseudo Code – Round Robin Algorithm . . . 210
D.3 Pseudo Code – Greedy Algorithm . . . 212
D.4 Pseudo Code – Attributes-Greedy Algorithm . . . 214
D.5 Pseudo Code – Attributes-Enhanced Algorithm . . . 217
D.6 Pseudo Code – Brute Force Algorithm . . . 218

List of Tables

2.1 Agent, Rank, and Training 16
2.2 Task Requirements 17
2.3 Possible Assignments 17

4.1 Attribute Values of Agent Types 57
4.2 Attribute and Capability Values of Agent Types 66
4.3 Attribute and Capability Values of Agent Types 75
4.4 Example Table 76

Acknowledgments

I would like to thank the people in my life who have contributed to this work. First, I am grateful for my major professor, Scott DeLoach, who has helped me tremendously throughout my work. His mentoring has helped me grow as a researcher and I appreciate the time he has taken to help me revise and improve this dissertation.

Second, I am grateful for the contributions and help from my colleagues. In this regard, I appreciate the input of Julie Adams and her research associate, Caroline Harriott, on the topic of human factors. I am grateful to Zhuang Rui, Jorge Valenzuela, and Joseph Lancaster for providing me with demonstrations on supervisory control.

Third, I am grateful for the support and encouragement from my friends, Scott Harmon, Edwin Rodríguez, and Matthew Miller. Their support and encouragement have kept me going to finish this dissertation.

Finally, I am grateful to Kansas State University for providing me with the opportunity to work on this dissertation.

Chapter 1

Introduction

As computing systems become more advanced, more is being expected out of them. Some of these expectations are (1) to perform complex tasks autonomously, (2) to be able to adapt to failures, and (3) to have closer integration with humans. Human integration with computing systems plays a part in achieving better autonomy and adaptation to failures. In this dissertation, human integration means two things: (1) humans have more control over computing systems and (2) humans are included as part of the decision-making process of a computing system.

Autonomy. Autonomy and adaptation to failure usually goes hand-in-hand. More autonomy in computing systems means more free time for the human operators; some of this free time can be used to manage multiple systems. In the field of Human-Computer Interaction (HCI), research is being done on interfaces that allow humans to efficiently interact with multiple systems simultaneously. Similarly, research in the field of Human-Robot Interaction (HRI) is borrowing ideas and concepts from HCI as the lines between robots and computing systems continue to blur. Robots are useful machines that allow tasks to be performed in places that would otherwise be too dangerous, difficult, or costly for humans. For example, (1) in urban search and rescue, robots can navigate in areas that

are too small for humans; (2) in hazardous waste cleanup, robots are more resilient when dealing with hazardous waste; and (3) in planetary exploration such as the ongoing Mars missions, robots require significantly less resources to explore the planet than do humans. In the early days of HRI, a single robot required multiple humans operators to control it. However, as technology continues to improve, the situation is starting to be reversed. There is a greater emphasis on one human operator controlling multiple robots due to reasons such as reducing cost [17].

In practical terms, autonomy is usually augmented by human supervision. Even for fully autonomous robots, human intervention is occasionally required to ensure continued operation. For example, the iRobot Roomba® Vacuuming Robot[1] is fully autonomous. Once turned on, the Roomba will start to perform its intended purpose such as vacuuming the floor while avoiding obstacles and recharging itself when necessary. However, there are times when the Roomba requires external help to continue. So, imagine a scenario where a cleaning crew is comprised of 20 Roombas and one human supervisor. The Roombas are deployed to vacuum rooms that are spread across multiple floors. It is inefficient for the human supervisor if each Roomba has to be checked periodically to ensure that it is still working correctly. It would be more effective if the Roombas would attempt to correct the situation by themselves first and then request assistance from the human supervisor via an HRI when they are unable to rectify the situation. Then the human supervisor could try to correct the situation remotely and, if failing to do so, proceed to the Roombas' locations to correct the situation manually. Thus, as the trend towards one human controlling multiple robots continues, there will be an increasing demand for more efficient and effective ways to interact with multiple robots.

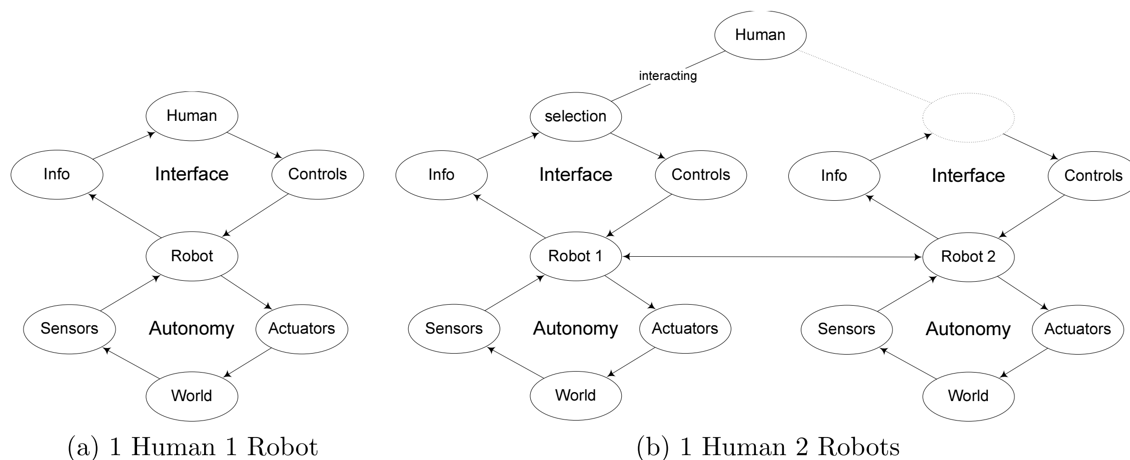
In HRI, Conway, Voltz, and Walker [16] define five categories of interaction: *teleoperation*, *shared control*, *traded control*, *supervisory control*, and *learning control*. In addition, Tsuji

[1] Further information can be obtained from http://store.irobot.com/

and Tanaka [99] add *impedance control* as the sixth type of interaction. *Teleoperation* means that a human operator assumes direct-control of a robot. *Shared control* means that a human operator controls a robot through high-level instructions such as "go here" and "pick up this object". *Traded control* is similar to *shared control* except that the human operator and the robot are in the same vicinity. Traditionally, *supervisory control* means that a human operator controls a robot through tasks. Tasks such as "search this area for explosives" require that robots to reason over them. *Learning control* means that a human operator controls a robot as if it was another human because the robot possesses an artificial intelligence that is comparable to human intelligence. *Impedance control* refers to robots that function as an extension to human actions such as a prosthetic limb or an exoskeleton.

Crandall *et al.* [18] propose that the interaction schemes from the six categories can be measured using *neglect time* and *interaction time*. *Neglect time* is "the expected amount of time a robot can be ignored before its performance drops below a threshold" and *interaction time* is "the expected amount of time that a human must interact with a robot to bring it to peak performance". With respect to controlling multiple robots, this measurement system implies that there is an upper bound to the number of robots that a single human can control. Even traditional supervisory control (as defined by Conway, Voltz, and Walker [16]), which has high neglect time and low interaction time, has an upper bound on the number of robots that a single human can control because the human is still interacting with each robot individually. A user study by Crandall and Cummings [17] suggests that the highest performance is somewhere between four and six robots. Figure 1.1a illustrates a single human operator controlling a single robot while Figure 1.1b illustrates the division of the human's attention between multiple robots. In Figure 1.1a, the interface loop is where the human interacts to control and receive information from the robot. And at the bottom, the autonomy loop is where the robot controls the actuators to interact with the world and uses the sensors to retrieve information about the world. In Figure 1.1b, there are two

robots. While the human operator is interacting with one robot, the other robot is left unattended (represented by the greyed-out oval in the interface loop). The human operator has to divide his/her attention between the robots. This form of supervisory control is sometimes referred to as *sequencing style* [44]. As long as the human is required to divide his/her attention between the robots, there is going to be a limit to the number of robots a human can control.



(a) 1 Human 1 Robot (b) 1 Human 2 Robots

Figure 1.1: Supervisory Control (Sequencing Style) [17]

Christoffersen and Woods [15] suggest that "the issue is not the level of autonomy or authority, but rather the degree of coordination". Johnson *et al.* [47] suggest that *coactivity* instead of autonomy should be the focus in systems where robots and humans are teammates. *Coactivity* is defined in three parts: (1) a group of participants is performing the same action (i.e., joint action), (2) a compulsion from the participants toward good teamwork, and (3) the abilities of participants allow external guidance (i.e., reciprocal action). They provide four reasons why autonomy is the wrong focal point.

1. The more autonomous a robot is, the less the robot depends on humans. However, the inverse happens to humans. Humans dependence on robots increases as robots gain more autonomy because the robots are in control of certain information and the

decisions that occur. This problem of the human becoming more dependent on the robot cannot be resolved with more autonomy.

2. Autonomy is not and cannot be perfect and thus will be prone to failure. Because autonomy is susceptible to failures, humans must intervene to correct the failure. Failures in autonomy cannot be solved with more autonomy, but can be solved with teamwork.

3. Some human activities cannot be replaced with autonomy without disrupting how the system works. This is because "humans cannot simply offload tasks to robots without incurring some coordination penalty" [47].

4. The last reason is humans themselves. Systems are typically built so that humans can benefit (in some way) from the systems. As such, humans want to understand the system but more importantly, humans want the ability to affect the system.

Thus, in order to control an arbitrary number of robots, a more scalable type of interaction is required. This leads to the first open question, *what new form of supervisory control can allow control over an arbitrary number of robots*. Ongoing research in the field of HRI is attempting to address the scalability limitation of the sequencing style of supervisory control. One particularly promising approach is to eliminate the need for the human to interact with the robots directly; there have been various attempts to increase the human-to-robots ratio [13, 64, 65, 69]. Instead of interacting with the robots directly, the human interacts with a system, which in turn handles the interaction with the robots. Figure 1.2 illustrates the evolving definition of supervisory control that deals with multiple robots as a group.

A particularly popular approach to controlling a group of robots is called the *playbook style* [44, 63], where the human selects a group of robots and the relevant "play" for that

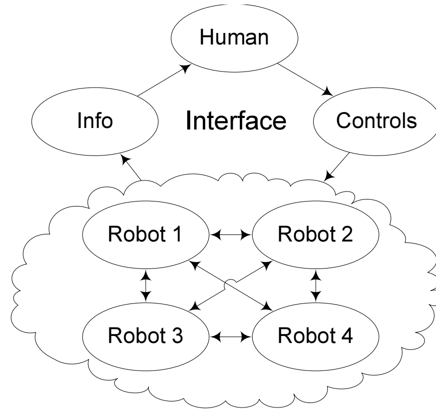


Figure 1.2: Supervisory Control

group. There are various implementations of the playbook style [44, 63, 64, 69]. My approach bears a similarity to playbook style in that control is over groups instead of individual robots. However, the means of control is different. In playbook style, "plays" are issued to groups. In my approach, high-level goals are issued to groups. A "play" is designed to achieve one or more objectives, which is equivalent to high-level goals. A "play" also typically specifies how many players there are and how each player behaves with regards to the objectives. In this respect, plays are more restrictive because the objectives are tightly coupled together with the behaviors specified to meet those objectives. My approach breaks the tight coupling by issuing high-level goals to groups while a separate mechanism decides how many members and the behaviors to achieve those goals.

Adaptation. Rising expectations for adaptive computing systems include the ability to automatically correct themselves in a fluid and dynamic environment (e.g., autonomic systems [49]). Multiagent concepts [11] are well-suited for developing adaptive systems. Russell and Norvig [79] define agents as able to perceive and act autonomously such that their actions are based on their own experiences rather than predefined knowledge. Multiagent Systems (MASs) exploit this behavior to self-correct; if one agent should fail, another agent can take over.

One approach in multiagent research is to leverage organizational concepts such as

agents, roles, and goals found in organizational models to produce Organization-based Multiagent System (OMAS). Some examples of such organizational models are Organization Model for Adaptive Computational Systems (OMACS) [26], Organizational Model for Normative Institutions (OMNI) [31], Organizations per Agents (OperA) [30], and HarmonIA [104]. By leveraging these organizational models, a general approach to adaptivity can be achieved through task allocations. Task allocations can be handled in a general manner because these models capture the necessary information to reallocate a task should an agent fail. Various research teams have applied organizational concepts in robotics, particularly in multirobot systems [9, 37, 41, 71, 89, 92, 95].

Another way of increasing a system's ability to adapt is by including humans as part of the system. Traditionally, humans have been considered as users of a computing system; humans are not typically considered as a factor during a system's decision making process. As computing systems continue to grow, the environments in which these systems operate sometimes involve humans. By including humans as a factor in these systems' decision making process, such systems are able to increase their adaptivity; tasks that can not be completed by the system due to failures can be allocated to humans for completion. There are two aspects involved when attempting to include humans as part of a system. First, designers must consider an interface to allow humans to interact with the system and vice versa. However, the actual requirements for such interfaces is beyond the scope of this dissertation. Second, an appropriate internal structure for a system to support humans so that the system can reason about humans and their abilities to complete tasks must be developed. This aspect significantly increases the complexity of such systems. One way to mitigate the increase in complexity is to represent humans in a general manner such that systems can reason over humans in an abstract way. Fortunately, organization-based models are well-suited to facilitate integration of humans because these models already provide a basic framework for representing humans; humans can be considered as agents. This leads

to the second open question: *what type of information about the humans should be captured that can lead to better allocation of tasks.*

1.1 Open Issues

Two open questions are identified above.

1. What new form of supervisory control can allow a human to control an arbitrary number of robots.

2. What type of information about humans should be captured that can lead to better allocation of tasks.

These two questions also show two different views to integrating humans with computing systems: (1) integration through human control over computing systems and (2) integration through inclusion of humans as part of a computing system's decision making process.

Firstly, facilitating better human control over an arbitrary number of robots requires an abstraction to capture and represent the robots as a group. In other words, the human needs to interact with a group and not individual robots. Figure 1.2 illustrates the idea of extending traditional supervisory control such that interactions are with a group of robots. The idea behind this new form of supervisory control is that since the human does not directly interact with the individual robots but rather a group of robots, the number of robots within that group should no longer be a limitation. Thus, a human should be able to control an arbitrary number of robots. Although supervisory control facilitates control over an arbitrary number of robots, there are still a number of challenges to overcome.

1. Investigate the types of failures that can occur. These failures are usually failures of individual robots that causes the group (in the current configuration) to fail in completing a task.

2. Define a mechanism to allow a group to autonomously change the group configuration so that the group can continue in completing the task without requiring human intervention.

3. Define a mechanism where a human can attempt to resolve the problem when/if the group fails to reorganize successfully.

4. Investigate the types of interactions that can occur between a human and a group.

Second, the types of information about humans that are relevant to task allocation need to be investigated. One type of information that is relevant for task allocation is human performance factors[2]. In the book by Wickens *et al.* [109], they examined and explained a large variety of human performance factors that are relevant to designing systems for human interaction. For instance, they explain the various human performance factors that affect the ability of a human to drive at night. The following are some of the human performance factors that impact the ability of a human to drive at night: the eyesight of the driver, the fatigue level of the driver, the reaction time of the driver, the color of objects, the luminosity of objects, the current weather conditions, the ambient lighting, and the speed of the car. For example, suppose there is one last task to deliver a package, it is snowing heavily, and there are two drivers available. Driver A has been driving for the past eight hours and is fatigued but driver B has only been driving for four hours and is less fatigued than driver A. Thus, it would be better to pick driver B who is less fatigued for the task. These human performance factors are not exclusive to any particular task and they can be classified into three categories: human-specific, task-specific, and environment-specific. There are a number of challenges to overcome in order to use human performance factors.

[2]The term “human factors” has multiple meanings. In order to distinguish between them, the term “human performance factors” refers to a specific definition where human factors are factors that affect the performance of an individual.

1. Define a means of capturing human performance factors so that they can be used by task allocation algorithms.
2. Define an appropriate mechanism so that a large number of human performance factors can be used at runtime for computing task allocations.

1.2 Approach

In the previous section, two general issues related to human integration with computing systems are highlighted: (1) how can a human interact with or control a group (organization) and (2) how to include and use information about humans as part of a system.

In my research, I have defined and developed a mechanism that allows humans to exercise supervisory control over a group of autonomous entities that are participating as part of an organization. The mechanism includes a definition of *organization control* (which is supervisory control over an organization) and a set of interactions that can occur when a human interacts with an organization. In order to eliminate or significantly reduce the need for the human to have direct interactions with the robots, I have developed a mechanism that interacts with the human. This mechanism converts the interactions with the human into individual interactions with the robots. In order words, the mechanism handles the individual interactions with the robots for the human. Furthermore, I have implemented a few of these interactions and will be evaluating them.

Next, Performance Moderator Functions (PMFs) [84] can be used to capture human performance factors, particularly Human Performance Moderator Functions (HPMFs) can be used to capture human-specific human performance factors. PMFs are a well-known and accepted approach to capturing human performance factors. In general, there are an enormous number of human performance factors [109]. Thus, when designing computing systems that include humans as part of the system (i.e., humans are considered as peers),

there can be a significant increase to the amount of information to be handled and the complexity of these systems can be overwhelming [81]. There is a need to discover an appropriate mechanism such that the complexity of systems that includes humans are not overwhelming. The complexities of including PMFs/HPMFs can be managed by leveraging Model Driven Engineering (MDE) [81]. By following the MDE approach, a runtime model (commonly referred to as *models@run.time* [7]) can be developed. This runtime model allows development of an adaptive mechanism that can autonomously perform task allocations. Furthermore, in the field of autonomous task allocation for multirobot systems, Parker [72] identified three paradigms for tackling the problem of task allocation. Two of the paradigms (the role-based organizational paradigm and the knowledge-based paradigm) tackle the problem of task allocations for heterogeneous robots in different ways. I followed the approach of OMACS, which combines both paradigms. However, the problem of task allocation is NP-hard [42], and thus, it is not realistic to expect optimal task allocations during runtime as general optimal task allocation algorithms would take too much time. In practical terms, greedy-based task allocation algorithms are often "good enough". Thus, the approach does not assume optimal task allocations from the algorithms but instead provides a mechanism to allow a human to modify the task allocations and uses element from both the role-based organizational and the knowledge-based paradigms.

This leads to the thesis statement, which is two-fold.

1.2.1 Thesis Statement

1. I will show that organization control coupled with an autonomous mechanism that has the ability to perform task allocations can significantly reduce the need for a human supervisor to interact with the individual robots.

2. In MASs that include humans as peers, I will show that task allocation algorithms that

are informed by HPMFs can lead to better results than uninformed task allocation algorithms.

1.3 Contributions

The following is a list of the contributions of my research.

- The definition of organization control and a set of interactions that can be used to implement organization control.
- The definition of an architecture that facilitates organization control. The architecture implements a mechanism that autonomously manages a group of robots and allows a human supervisor to exercise supervisory control over the organization through interactions with the organization instead of the individual robots.
- The implementation and demonstrations of several organization control interactions.
- The definition of a runtime model that captures HPMFs so that autonomous mechanisms can reason about humans with respect to their ability to perform tasks.
- The implementation and demonstrations of a runtime model that can lead to better task allocations for systems comprised entirely of humans or a mix of humans and robots in bulk and incremental task allocations.

1.4 Overview

The rest of this dissertation is organized as follows. Chapter 2 describes a scenario that functions as the motivating example for the work in this dissertation. Chapter 3 highlights the foundational work on which this dissertation is based. Chapter 4 describes the runtime model. Chapter 5 defines organization control, the architecture that facilitates organization

control, and demonstrations of several organization control interactions. Chapter 6 discusses research that is similar to the work in this dissertation. And Chapter 7 concludes by summarizing the work and contributions of this dissertation and highlighting some future work.

Chapter 2

Motivating Example

This chapter describes a scenario that serves as a motivating example for the work in my research. Section 2.1 describes the scenario and Section 2.2 follows through the scenario with an example script.

2.1 Scenario

An earthquake of magnitude 9.1 struck Research Lab A, which is a six-story building with four more levels underground. Research Lab A conducts research in various fields, some of which involves hazardous materials. Some parts of the building collapsed from the quake, resulting in large piles of rubble. Some parts of the building that are still intact are structurally fragile and could collapse anytime and cause further damage. The status of the underground levels are unknown but is assumed to be just as bad. An emergency response team (Human Responders and Survey Robots) is dispatched to the scene. The emergency response team is lead by a commander and is comprised of three groups that perform one of the following three functions: (1) survey, (2) hazard identification, and (3) victim rescue. The survey, hazard identification, and victim rescue groups consist of

multiple teams. A team contains one human and one robot. The robots are built for exactly one of the three functions of the emergency response team and they are not interchangeable across groups. The humans, however, could have cross training in the three functions and can be interchangeable.

All robots from the emergency response team are equipped with the following capabilities: (1) wireless communications; (2) GPS (for outdoor use), which has an accuracy up to 10 cm; (3) gyroscope (for indoor use), which can be used to infer positioning when GPS data is unavailable such as inside a building; (4) track-based wheels; and (5) navigation algorithms.

All human members are given an Android[1] device. The Android device is the primary means of interaction with the robots, the emergency response team system, and other human members. The following are actions taken through the Android device: (1) verbal and visual communications with other human members (either in one-to-one mode or conference mode), (2) looking up the state of their robot partner, (3) correcting inconsistencies in their robot partner, (4) receiving notifications from their robot partner, and (5) receiving new instructions and/or objectives from the emergency response team commander.

For the purpose of completeness, Appendix B provides further details on the emergency response team. The next section describes an example situation for the emergency response team.

2.2 Example Script

This section describes an example script to illustrate the situations and cases that I address in this dissertation, as well as serving as a central example for explaining concepts and ideas throughout this dissertation. In the example script, there are three robots in the

[1] Further information about Android is available at http://www.android.com/

survey group, two robots in the hazard identification group, and one robot in the victim rescue group. There are three human members (excluding the commander) in the emergency response team. The human members are John, Rob, and Stan. They have different ranks (which is equivalent to experience) and have gone through different training regimes (which is equivalent to skill sets).

John has just finish basic training and promoted to the rank of Emergency Response Officer (ERO), which is the basic starting rank in the emergency response team. Rob has the rank of Sergeant, which is the next rank up from ERO. In addition to basic training, Rob has also completed the Biological Agents Module (BAM) training. Lieutenant Stan has also completed BAM training. Unfortunately, none of them has gone through the Structural Hazards Awareness Module (SHAM) training. Table 2.1 shows the tabular form of the three human members with their rank and training.

	Rank	Training
John	ERO	Basic
Rob	Sergeant	Basic, BAM
Stan	Lieutenant	Basic, BAM

Table 2.1: Agent, Rank, and Training

There are fours areas ($A1, A2, A3, A4$) that need to be surveyed. The area $A1$ is the area around the part of the building that collapsed. There is rubble but nothing else dangerous so that anyone with basic training can survey $A1$. The area $A2$ is the ground floor entrance of the building, which is at the part of the building that is still intact but structurally fragile and could collapse anytime. So, to avoid further casualties, surveying $A2$ would require someone who has experience (e.g., at least a Sergeant rank) and someone who has been through SHAM training. The area $A3$ is the parking garage located in the first floor basement. The parking garage was built with sturdy materials and is moderately safe for surveying. In fact, surveying $A3$ would provide crucial SHAM experience such that someone surveying $A3$ would acquire a field training that is equivalent to SHAM training. Because

$A3$ provides an invaluable training experience, it is preferable that someone with experience (e.g., at least Sergeant rank) be selected to survey $A3$. The area $A4$ is where the biological labs are located. The structural integrity of the building in $A4$ is weak but still standing. However, the sensors that are still functioning indicate biological contamination. So, there is a high chance of biological contamination. Therefore, the team going to $A4$ will be both surveying and analyzing $A4$ for biological contamination.

The above description results in five tasks: survey $A1$, survey $A2$, survey $A3$, survey $A4$, and hazards identification $A4$. Table 2.2 shows the requirements and gains of performing the task in terms of rank and training.

	Rank	Training	
	Requires	Requires	Gains
Survey $A1$	Any	Basic	–
Survey $A2$	At least Lieutenant	Basic, SHAM	–
Survey $A3$	At least Sergeant	Basic	SHAM
Survey $A4$	Any	Basic	–
Hazards ID $A4$	Any	BAM	–

Table 2.2: Task Requirements

Based on the requirements of the tasks and the human members available, there are a number of possible initial assignments for the human members. Table 2.3 shows the possible initial assignments for the three human members of the emergency response team. At first, no human member can be assigned to survey $A2$ because the task requires someone who has the necessary SHAM skills.

	Possible Assignments
John	Survey $A1$, $A4$
Rob	Survey $A1$, $A3$, $A4$, Hazards ID $A4$
Stan	Survey $A1$, $A3$, $A4$, Hazards ID $A4$

Table 2.3: Possible Assignments

However, not all the possible assignments are viable for the long term goal. For instance, if Rob is assigned to survey $A3$, he would gain the equivalent SHAM skill upon completing

that task; but then there are no human members who can be assigned to survey $A2$. Therefore, the commander should pick Stan instead of Rob for the task of surveying $A3$.

Suppose that John is given the task of surveying $A1$ and is paired with a robot partner named Surveyor 1. At the start, John discusses with Surveyor 1 about how to get to $A1$. Since this is John's first deployment and Surveyor 1 has experience, John decides to let Surveyor 1 lead the way to $A1$. John notices on the Android device that his map has been updated to show the path that Surveyor 1 will take to reach $A1$. As John follows Surveyor 1 to $A1$, he also notices that his map is being updated by Surveyor 1 to indicate paths and obstructions. Halfway through the path, John sees a quicker path to $A1$. John informs Surveyor 1 of this preferred path using his Android device. Surveyor 1 corrects itself and navigates the updated path to $A1$.

Upon reaching $A1$, Surveyor 1 begins its systematic search of $A1$ and John begins to visually survey the area and updates his map on the Android device. As John is updating his map, he receives a request from Surveyor 1 to look at subarea $A1a$, where Surveyor 1 could not search. John proceeds to $A1a$ and searches $A1a$. As John nears completion of searching $A1a$, Surveyor 1 finds a survivor. Surveyor 1 notices that John is busy searching $A1a$ but is nearly done, so Surveyor 1 decides to wait until John completes his search. When John finishes his search, Surveyor 1 informs John of the survivor. John proceeds immediately to the survivor's location.

Once John arrives at the survivor's location, Surveyor 1 resumes its search. John determines that the survivor is mobile and decides to escort the survivor back to base. The commander notices that John is escorting the survivor back and assigns Rob to replace John. Rob heads to $A1$, following the path taken earlier as indicated on his map. As Rob is making his way to join Surveyor 1, Surveyor 1 discovers an unconscious survivor. Surveyor 1 informs the commander of an unconscious survivor. The commander dispatches Stan and the Stretcher 1 to the location. Surveyor 1 notices that John is no longer available and

that Rob is on his way. Surveyor 1 stays with the survivor until Rob arrives on the scene. When Rob arrives at $A1$, he receives a notification from Surveyor 1 that an unconscious survivor is found. Rob proceeds immediately to the survivor's location. When Rob reaches the location, Surveyor 1 resumes its search. Rob realizes that a victim rescue team is already on the way and waits for the team to arrive.

Stan and Stretcher 1 arrive at the unconscious survivor's location. Stretcher 1 prepares itself to take the survivor on the stretcher. Stan and Rob prepare the survivor to be moved onto the stretcher. While the survivor is being prepared to be evacuated, Surveyor 1 completes its search and informs Rob. Surveyor 1 realizes that Rob is busy preparing the survivor for evacuation and decides to join up with Rob. When Surveyor 1 reaches Rob's location, it enters standby mode to conserve power until Rob is ready. When the survivor is secured to the stretcher, Stan and Stretcher 1 proceed back to base. Rob wakes up Surveyor 1 and the team proceeds back to base.

2.3 Highlights

The example script (Section 2.2) highlights situations where controls (either by the commander or the human partner) and the interactions involved. The following is a list of controls and interactions. Only the first three are addressed in my research.

- In the example script, the commander makes the decision on who to assign to tasks. This decision can also be made autonomously by a task allocation algorithm. In my research, I show how such task allocations can be done autonomously.
- The commander, who has *supervisory control* over the system, is able to change parameters of the system should the need arise such as replacing John with Rob when Rob is escorting a survivor back. In my research, I will show how a human

supervisor can exercise this type of control and also how this type of control can also be done autonomously.

- In the example script, the parameters of a robot can be modified by a human such as modifying the robot's navigation path. In my research, I show how this type of control can be done.
- In the example script, the robot modifies its behavior such as waiting for John to finish searching before informing John of a survivor.

2.4 Summary

This chapter describes a scenario that serves as the motivation for the work in my research. The example script highlights some of interactions and control that are enabled by the work in this dissertation. In addition, the example (Section 2.2) also serves to help explain concepts and ideas in later parts of this dissertation. The next chapter (Chapter 3) covers the necessary background information that forms the basis for the work in this dissertation.

Chapter 3

Background

This chapter highlights key background areas required to understand the work in my research. Section 3.1 describes the OMACS model. OMACS is the basis for the Chazm Model (CzM) (which is presented in Chapter 4). Section 3.2 describe the Goal Model for Dynamic Systems (GMoDS) model. GMoDS is the goal model that I am extending to allow goal modifications (Section 5.3.2). Section 3.3 describes the Organization-based Multiagent System Engineering (O-MaSE) methodology that I used in building the system described in Section 5.2.1. Section 3.4 describes human factors and highlights the mathematical concepts to explain and capture human factors. Section 3.5 describes the various aspects of HRIs.

3.1 Organization Model for Adaptive Computational Systems (OMACS)

This section describes OMACS [26]. OMACS forms the foundation on which the CzM model is built. OMACS is a model that captures the knowledge required to allow a team of autonomous agents to adapt to failures or changing goals. As shown in Figure 3.1, an OMACS *organization* consists of *goals*, *roles*, *agents*, and *capabilities*. *Goals* are high-

level descriptions of what the system is supposed to accomplish [78]. *Roles* are high-level specifications on how to achieve specific goals. *Agents* are autonomous entities that can perceive and act within their environment [78]. *Capabilities* represent the notion of an agent's ability to perceive and act on its environment.

Figure 3.1: OMACS Model

These entities are related to one another via a set of functions: *achieves*, *requires*, *possesses*, *capable*, and *potential*. The *achieves* function defines the effectiveness $[0.0, 1.0]$ of a role in achieving a goal, where 0.0 means that the role is unable to achieve the goal. The *requires* function defines the capabilities that a role needs in order for agents to carry out the role's behavior. The *possesses* function defines the effectiveness $[0.0, 1.0]$ of an agent's capabilities, where 0.0 means that the capability is broken or non-existent. The *capable* function specifies how well $[0.0, 1.0]$ an agent can perform a role, where 0.0 means that the agent is unable to perform the role. The *potential* function defines how well $[0.0, 1.0]$ an agent can perform a role to achieve a goal, where 0.0 means that the agent is unable to perform the role to achieve the goal.

There is a user definable function called the `rcf` function. The `rcf` function computes a score $[0.0, 1.0]$ that indicates how well an agent (with its current capabilities) is able to perform a role. The `rcf` function is user definable because it is not possible to represent all ways of computing a score through one general function. An example of the `rcf` function

is shown in Equation 3.1, which uses the *requires* and *possesses* functions. The example `rcf` function ensures that a given agent possesses the required capabilities of a given role. Because the `rcf` function is user definable, a custom `rcf` function can specify a minimum competency level for the capabilities that agents must have in order to be considered eligible for performing the role. In OMACS, because of the way the `rcf` function is defined, the *capable* function is the same as the `rcf` function: $\mathit{rcf}(a, r) = \mathit{capable}(a, r)$. The *potential* function combines the score from the `rcf` function with the score from the *achieves* function: $\mathit{potential}(a, r, g) = \mathit{rcf}(a, r) \times \mathit{achieves}(r, g)$.

$$\mathrm{rcf}(a, r) = \sqrt[|\{c|(r,c)\in \mathrm{requires}\}|]{\prod_{c\in\{c|(r,c)\in \mathrm{requires}\}} \mathrm{possesses}(a, c)} \tag{3.1}$$

OMACS-based systems use the *potential* function to autonomously make assignments[1]. An assignment is a tuple consisting of one goal, one role, and one agent. As the capabilities of agents change throughout the course of a system's execution, these changes are reflected through the *possesses* function, which then is also reflected by the `rcf` function, and finally by the *potential* function. Should an agent reach a point where it is no longer capable of performing a role, the `rcf` function would return a score of 0.0. This would, in turn, cause the *potential* function to return a score of 0.0, and if that agent is still assigned to perform the associated role to achieve the associated goal, a reorganization must occur at this point to replace the failed agent. In this manner, an OMACS-based system adapts to failures.

3.2 Goal Model for Dynamic Systems (GMoDS)

This section describes the formalization behind GMoDS [25, 66]. In this dissertation, GMoDS is extended to allow goal modifications to occur. The extension reuses existing definitions from the GMoDS and so, those definitions will be reiterated again. In OMACS,

[1]Making assignments is synonymous with task allocation.

the set of goals simply represents the current goals that the organization is actively pursuing. A sophisticated model is required to represent a more complex set of requirements such as alternative goals, goal sequencing, and situational goals.

In GMoDS, a system's requirements is captured as a single goal tree. The top-level or overall goal is decomposed into subgoals that follows the classic AND/OR goal decomposition [100]. If all subgoals must be achieved to achieve the parent goal, then the parent goal is an AND-goal. Conversely, a goal is an OR-goal if that goal is achieved when any of its subgoals are achieved. At the lowest level of the goal tree are the leaf goals (goals without subgoals), which are goals that are used by OMACS-based systems for making assignments.

GMoDS consists of two parts: (1) a specification model that captures the generic requirements of a system and (2) an instance model that tracks the progression of a system towards achieving the overall goal. The classic AND/OR goal tree provides the ability to track progression in achieving a system's overall goal. In addition to the classic AND/OR goal tree, the GMoDS provides two additional features. First, GMoDS provides the ability to systematically track incremental progress towards achieving the system's overall goal (i.e., a sequential ordering for achieving goals) through the *precedes* relation. For instance, if goal A *precedes* goal B, then goal A must be completed first before goal B can be attempted. Second, GMoDS provides the ability to dynamically adapt to variations in system parameters (capturing events that cause the creation of new goals or removal of existing goals) through the *triggers* relation. For instance, if goal A *triggers* goal B on event E, then while in the pursuit of goal A, goal B is created every time event E occurs.

3.2.1 Specification Model

The specification model is where the *precedes* and *triggers* relations are specified. Starting with the definition of a goal in the specification model. A goal in the specification model is

known as a *specification goal*.

Definition 3.1. A *specification goal* is a goal type. For example, the *survey A1*, *survey A2*, *survey A3*, and *survey A4* goals (from Chapter 2) are from the same goal type: *survey*. Thus, G_{S} is the set of all specification goals.

Because all specification goals are captured in a tree-like structure, there are four functions that describe the relationships among specification goals in that tree-like structure. The first function is the *parent* function that returns the parent specification goal. In the GMoDS, all specification goals except the overall specification goal has exactly one parent.

Function 3.1. *parent* $: G_{\mathrm{S}} \mapsto G_{\mathrm{S}}$. Given two specification goals, g_1 and g_2, if g_1 is a subgoal of g_2 then $\mathrm{parent}(g_1) = g_2$, which means that g_2 is a parent of g_1.

The *ancestor* function expands on the *parent* function. The *ancestor* function is the transitive closure of $\mathrm{parent}(g)^{+}$, where $g \in G_{\mathrm{S}}$. The *ancestor* function returns a set of specification goals that are the ancestors of the given specification goal.

Function 3.2. *ancestor* $: G_{\mathrm{S}} \mapsto \mathbb{P}(G_{\mathrm{S}})$. The *ancestor* function is defined recursively as $\mathrm{ancestor}(g) = \bigcup_{g' \in \mathrm{parent}(g)} \mathrm{ancestor}(g')$, where $g \in G_{\mathrm{S}}$.

The next function is the *children* function that returns the subgoals of a given specification goal.

Function 3.3. *children* $: G_{\mathrm{S}} \mapsto \mathbb{P}(G_{\mathrm{S}})$. Given three specification goals, g_1, g_2, and g_3, if g_2 and g_3 are subgoals of g_1 then $\mathrm{children}(g_1) = \{g_2, g_3\}$, which means that g_2 and g_3 are the children of g_1.

Similar to the *ancestor* function, the *descendant* function expands on the *children* function. The *descendant* function is the transitive closure of $\mathrm{children}(g)^{+}$, where $g \in G_{\mathrm{S}}$.

Function 3.4. *descendant* $: G_{\mathrm{S}} \mapsto \mathbb{P}(G_{\mathrm{S}})$. The *descendant* function is defined recursively as $\mathrm{descendant}(g) = \bigcup_{g' \in \mathrm{children}(g)} \mathrm{descendant}(g')$, where $g \in G_{\mathrm{S}}$.

The next definition is the *triggers* relation.

Definition 3.2. A *trigger* is a tuple of $\langle E, G_{\mathrm{S}}, G_{\mathrm{S}} \rangle$. Given an event e and the specification goals g_1 and g_2, a trigger (e, g_1, g_2) means that g_1 can trigger g_2 if event e occurs. Thus, T is the set of all triggers.

3.2.2 Instance Model

The following are definitions of the instance model. First, the definition of a goal in the instance model. A goal in the instance model is known as an *instance goal*.

Definition 3.3. An *instance goal* is an instance of a specification goal. For example, the *survey A1*, *survey A2*, *survey A3*, and *survey A4* goals (from Chapter 2) are instance goals. Thus, G_{I} is the set of all instance goals.

Because all instance goals are associated with a specification goal, the *spec* function defines the relations between instance goals and specification goals.

Function 3.5. $\mathit{spec} : G_{\mathrm{I}} \mapsto G_{\mathrm{S}}$. For example, the *survey A1* goal (from Chapter 2) comes from the specification goal *survey*. Thus, $\mathrm{spec}\left(\mathit{survey\ A1}\right) = \mathit{survey}$.

Similar to the specification model, all goals in the instance model are stored in a tree-like structure. The next four functions describe the relationships among instance goals.

Function 3.6. $\mathit{parent} : G_{\mathrm{I}} \mapsto G_{\mathrm{I}}$. Given two instance goals, g_1 and g_2, if g_1 is a subgoal of g_2 then $\mathrm{parent}\left(g_1\right) = g_2$, which means that g_2 is a parent of g_1.

The *ancestor* function expands on the *parent* function. The *ancestor* function is the transitive closure of $\mathrm{parent}\left(g\right)^{+}$, where $g \in G_{\mathrm{I}}$. The *ancestor* function returns a set of instance goals that are the ancestors of the given instance goal.

Function 3.7. $\mathit{ancestor} : G_{\mathrm{I}} \mapsto \mathbb{P}\left(G_{\mathrm{I}}\right)$. The *ancestor* function is defined recursively as $\mathrm{ancestor}\left(g\right) = \bigcup_{g' \in \mathrm{parent}(g)} \mathrm{ancestor}\left(g'\right)$, where $g \in G_{\mathrm{I}}$.

The next function is the *children* function that returns the subgoals of a given instance goal.

Function 3.8. *children* : $G_{\mathrm{I}} \mapsto \mathbb{P}(G_{\mathrm{I}})$. Given three instance goals, g_1, g_2, and g_3, if g_2 and g_3 are subgoals of g_1 then $\mathrm{children}(g_1) = \{g_2, g_3\}$, which means that g_2 and g_3 are the children of g_1.

Similar to the *ancestor* function, the *descendant* function expands on the *children* function. The *descendant* function is the transitive closure of $\mathrm{children}(g)^{+}$, where $g \in G_{\mathrm{I}}$.

Function 3.9. *descendant* : $G_{\mathrm{I}} \mapsto \mathbb{P}(G_{\mathrm{I}})$. The *descendant* function is defined recursively as $\mathrm{descendant}(g) = \bigcup_{g' \in \mathrm{children}(g)} \mathrm{descendant}(g')$, where $g \in G_{\mathrm{I}}$.

For further details on the GMoDS, look at thesis [66] or the journal paper [25].

3.3 Organization-based Multiagent System Engineering (O-MaSE)

This section describes the O-MaSE [24, 40] methodology framework, which I used to build the system described in Chapter 5. The O-MaSE methodology is an organization-based agent-oriented design process that expands on existing MAS methodologies, particularly from an earlier work Multiagent Systems Engineering (MaSE) [28]. Furthermore, the O-MaSE methodology provides the necessary key concepts to designing and implementing MASs and allows designers to define an OMACS-compliant development process. The O-MaSE methodology is based on the SPEM 2.0 [45] and the O-MaSE meta-models. Figure 3.2 illustrates the O-MaSE methodology framework, where the meta-model, the fragments, and the process construction guidelines are the main components of the O-MaSE methodology framework. In addition, the O-MaSE methodology is supported by agentTool

III (aT^3) [38, 39]. A process engineer starts by selecting the necessary fragments and then creates the associated processes while keeping to the defined guidelines. Each fragment is an instance of an element from the SPEM meta-model, which is defined in terms of the O-MaSE meta-model.

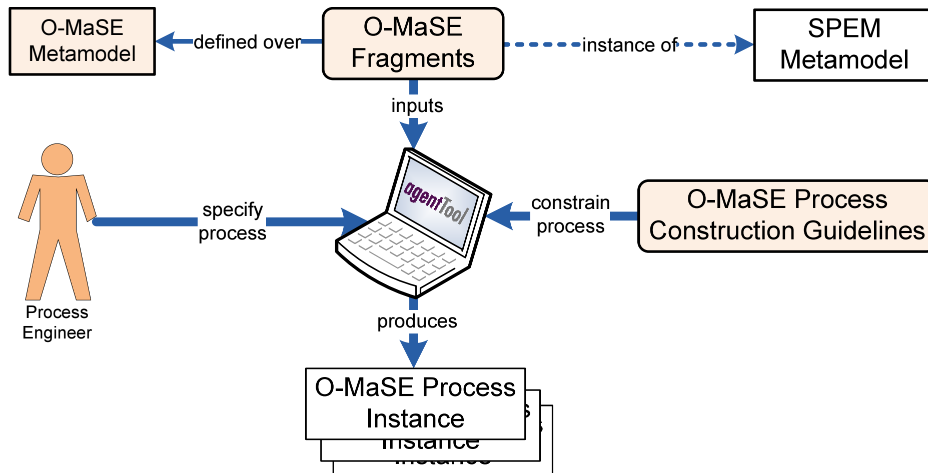


Figure 3.2: O-MaSE Methodology Framework [24]

The O-MaSE meta-model is derived from the OMACS model and includes additional concepts and relationships to define MASs. Figure 3.3 shows the O-MaSE meta-model (shaded rectangles are elements that correspond to elements from OMACS). An *organization* (in O-MaSE) consists of six entities: *goals*, *roles*, *agents*, a *domain model*, and *policies*. *Goals*, *roles*, and *agents* share the same definitions from OMACS. The *domain model* captures the necessary information about the environment in which the agents will operate. *Policies* constrain an organization by limiting how the organization behaves in certain situations.

In order for agents to operate within an environment, agents possesses a set of *capabilities*. *Capabilities* can be defined as (1) a composition of multiple *capabilities*, (2) a set of *actions* that the *capability* can carry out, and (3) a set of *plans* that defines a sequence of *actions* to use.

To capture the notion of a hierarchy in organizations, the O-MaSE meta-model defines the *organizational agents*. *Organizational agents* are organizations that acts as agent within

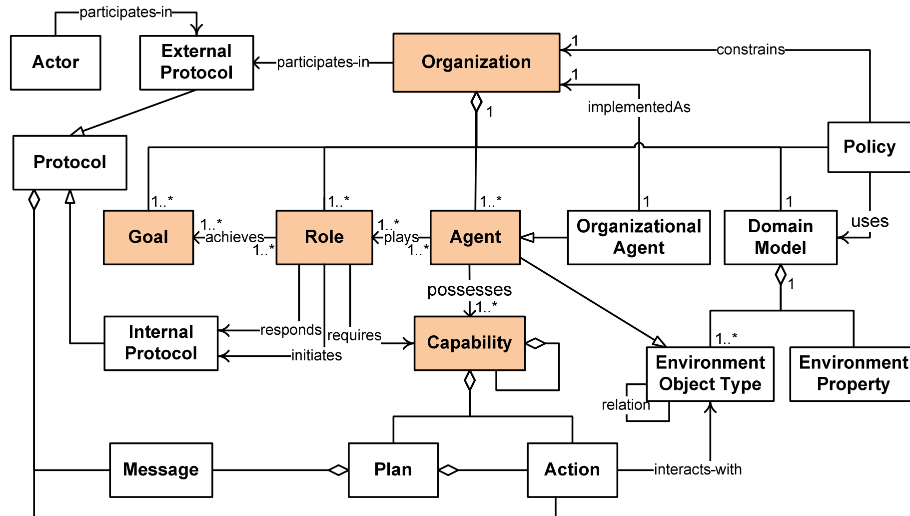


Figure 3.3: O-MaSE Meta-Model [24]

another organization. Just like agents, organizational agents possesses capabilities and are capable of playing roles.

In order for communications to occur, the O-MaSE meta-model define *protocols* for communications among roles and also for communications between external entities (defined as *actors*) and the organization. Thus, there are two types of *protocols*: *internal protocols* and *external protocols*. A *protocol* can be defined in terms of either a sequence of *messages* or *actions*.

In O-MaSE, there are various types of tasks. Only a subset of the tasks are covered here because the end result of those tasks are models that are used in Chapter 5. For more detailed information on O-MaSE, look at the paper by DeLoach and García-Ojeda [24].

The first two tasks, Model Goals and Refine Goals, result in a goal model as the work product. The Model Goals task transforms the system requirements into goals. A common approach to modeling goals is the classic AND/OR goal decomposition [100]. Once the initial goal model is done, then Refine Goals task refines the initial goal model so that dynamic aspects of the system are captured such as a sequential order to achieving goals, events that cause new goals to be created, and defining parameters that describe the state

of goals. An example of a goal model after the Refine Goals task is completed is GMoDS [66].

The tasks Model Roles and Define Roles result in a role model as the work product. The purpose of the Model Roles task is to identify the roles, the interactions that occur among roles, and the leaf goals that the roles can achieve. The Define Roles task defines the capabilities that agents must have in order to play a role. In addition, the Define Roles task also specifies the behavior for how agents play the roles.

The Model Agent Classes task results in an agent model as the work product. The purpose of the Model Agent Classes task is to identify the types of agents that can exist in the system. The types of agent can be defined in one of two ways. The first way is to define the agent types in terms of the roles that the agents may play. The second way is to define the agent types in terms of capabilities that the agents have, which corresponds to the required capabilities defined for roles.

The Model Capabilities task results in a capability model as the work product. The purpose of the Model Capabilities task is to define the capabilities of agents in terms of either actions or plans.

For further information on the other aspects of O-MaSE, look at the paper by DeLoach and García-Ojeda [24].

3.4 Performance Moderator Functions (PMFs)

This section explains human factors and how the study of human factors has lead to the development of PMFs. The goal of human factors (as defined by Wickens *et al.* [109]) is to facilitate better human interaction with systems so that (1) performance is enhanced, (2) safety is increased, and (3) user satisfaction is increased. The study of human factors is multidisciplinary as it encompasses (1) understanding of the human mind and how the

human brain processes information, (2) understanding the physical aspects of the human body, and (3) understanding how the human brain and body work with systems. The study of human factors is also known as human factors sciences. Furthermore, human factors engineering is the application of the knowledge gained from the study of human factors. In their book, Wickens *et al.* [109] explore a number of design principles and methodologies to designing better systems.

For example, the current design of road signs is a result extensive study and experimentation of human factors. There are various aspects (such as shape, color, and manufacturing materials) to designing a good road sign. One of the human factors included in the study is the human eye. A human eye can perceive light in the wavelength ranging from 400 nanometers (i.e., the color violet) to 700 nanometers (i.e., the color red). However, the human eye often perceive multiple light sources at the same time. A light source can be characterized by *hue*, *saturation*, and *brightness*. *Hue* is typically defined in terms of the three primary colors: red, blue, and green. *Saturation* is whether a light source is diluted (unsaturated) or undiluted (saturated) by another color such as gray. *Brightness* is the intensity of the light source. For instance, for a human to see a road sign at night, the formula $R = L/I$ [109] (where R is *reflectance*, L is *luminance*, and I is *illuminance*) approximates the brightness the road sign. *Reflectance* is a percentage of how much light is reflected. *Illuminance* refers to how much light the road sign is getting, which is typically the head lights of the car. *Luminance* refers to how much light is reflected off the road sign. But brightness is not all there is to designing good road signs. Another important human factor is also *contrast*. *Contrast* helps in discerning the different types of road signs, which can be measured as $C = (L - D)/(L + D)$ [109], where C is contrast, L is the luminance of the light areas, and D is the luminance of the dark areas. The ability of a human eye to discern contrast is known as *contrast sensitivity*, which can be measured as $S = 1/T$ [109], where S is contrast sensitivity and T is the minimum amount of contrast that can be

detected.

Human factors play an important role in the development of simulations that model human behavioral and cognitive processes. For example, the Department of Defense (DoD) Modeling & Simulation Coordination Office (M&S CO)[2] creates realistic and complex virtual worlds for training soldiers. These virtual worlds are populated with virtual combatants that are highly realistic. Human factors play a part in guiding the behaviors of these virtual combatants. However, the literature in human factors is vast and extracting these factors into a form that is suitable for implementation often require expertise in the associated field. That is where research in PMFs bridges the gap between the literature and implementation. PMFs are the quantified form of human factors. Developers can use PMFs in their implementations without requiring expertise in the associated fields.

PMFs [84, 85] indicate the impact of internal and external human factors on human performance. Examples of internal human factors are fatigue level, reaction time, and mental acuity. Examples of external human factors are noise level, lighting, and task time. In addition, PMFs are able to capture impact of personality on performance such as emotion, cultural background, and biases. Furthermore, PMFs quantify performance differences between two humans such as intelligence, skill, and motivation. In other words, PMFs capture the relationship between performance moderators and the level of performance in the form of dose-response (or exposure-response). As the dose (or exposure) increases or decreases, PMFs indicate the change in the level of performance.

An example of a PMF is shown in Equation 3.2 [84] for computing how people make decisions given two risky prospects, where x represents the gain for the first prospect with probability p and y represents the gain for the second prospect with probability q.

[2] More information can be obtained from the website http://www.msco.mil/.

$$V(x,p;y,q) = \begin{cases} \pi(p)v(x) + \pi(q)v(y) & \text{if } (p+q<1) \vee (x \geq 0 \geq y) \vee (x \leq 0 \leq y) \\ v(y) + \pi(p)[v(x) - v(y)] & \text{if } (p+q=1) \wedge ((x>y>0) \vee (x<y<0)) \end{cases} \tag{3.2}$$

For instance, the first prospect could be winning a small amount of money (x) with a high probability (p) and the second prospect could be winning a large amount of money (y) with a low probability (q).

3.5 Human-Robot Interaction (HRI)

This section describes four key aspects to HRI. These aspects are not independent of each other.

1. The human should be able to perceive the environment in which the robot is operating (also known as *situation awareness* [35]).

2. The human should be able to convey their intentions to the robot, preferably in a natural way so as to reduce the *mental workload* [69] on the human.

3. The robot should be able to comprehend the intentions of the human (e.g., *perspective taking* [97] is one such approach).

4. The robot should be able to provide humans with useful information and not just data [15].

First, the roles that humans play when they are interacting with robots are described in Section 3.5.1. The roles that humans adopt is a contributing factor to the types of interactions that occur between humans and robots. Next, the various interaction schemes

are described in Section 3.5.2. And finally, the various levels of automation of robots are described in Section 3.5.3.

3.5.1 Human Roles

When humans interact with robots, there are a number of roles that humans can play. Scholtz [82] defined five types of roles: *supervisor*, *operator*, *mechanics*, *peer*, and *bystander*. Furthermore, Scholtz, Antonishek, and Young [83] suggest that when humans are playing these roles, they are not played in a mutually exclusive manner. Sometimes, when a human is playing the role of a *supervisor*, he/she may also play the role of an *operator*. Each of the five roles (as defined by Scholtz [82]) have different requirements and expectations on HRIs. The following describes the requirements and expectations of each role.

Supervisors – when assuming the role of a *supervisor*, the interactions that occur between the human and the robot are similar to the interactions that occur between humans.

Operators – when assuming the role of an *operator*, the human is typically in direct control of the robot. The *operator* is allowed access to the internal state of the robot as well as direct-control of the robot's action.

Mechanics – humans take on the role of a *mechanic* when they are required to deal with the physical aspects of the robots such as adjusting the angle of a camera.

Peers – when assuming the role of a *peer*, humans and robots coexist as teammates. As teammates, both humans and robots are required to work together to achieve some common goals, which is usually given by the *supervisor*.

Bystanders – when assuming the role of a *bystander*, humans have little or no direct interactions with the robots. Typically, robots do not work with *bystanders*. A *bystander* can indirectly interact with the robot such as standing in the path that

the robot is navigating; and typically, the robot reacts by navigating around the *bystander*.

3.5.2 Interaction Schemes

Conway, Voltz, and Walker [16] defined five categories of HRIs: *teleoperation*, *shared control*, *traded control*, *supervisory control*, and *learning control*. In addition, Tsuji and Tanaka [99] defined *impedance control* as the sixth category of HRIs.

Teleoperation – the human has direct-control of the robot. Typically, the human controls the robot from a remote location. The robot has little autonomy (if there is any autonomy at all in the robot) because the human controls the robot using the lowest-level operations. If there is some autonomy in the robot, the human still has direct control of the robot, but the robot is given some autonomy such as preventing damage to itself by not allowing the human to collide with objects; this level of autonomy is often called *safe mode* [10] .

In order to permit the human *teleoperation* control over the robot, the human would have to be aware of the environment in which the robot is located (*situation awareness* [35]). Endsley [35] defined three levels of *situation awareness*: *perception* (level 1), *comprehension* (level 2), and *projection* (level 3). *Perception* is the ability of the interface to provide the necessary information so that humans are able to perform their function. *Comprehension* is how the interface is able to combine information and how the interface interprets the information. *Projection* is the ability to predict the future events based on current events and the state of the environment.

A great deal of research has been done on the area of perception (typically referred to as *telepresence*). The goal of *telepresence* research is to create an interface such that humans would feel as if they are actually present in the environment of the remote robots. However,

according to Woods *et al.* [110], accomplishing this goal is not easy for many reasons. One of the problems is the inability of humans to perceive the scale of objects [110] through video feeds or images. Kanduri *et al.* [48] describe the problem of height perception when humans attempt to navigate robots in remote environments using a single video feed (monoscopic vision). Kanduri *et al.* [48] proposed a solution in the form of post-image processing by performing horizon analysis. However, horizon analysis was only shown to work reliably when the terrain is generally flat. Hughes and Lewis [46] proposed another solution that used two cameras to provide stereoscopic vision. However, the amount of bandwidth required to provide two video feeds is currently impractical especially since providing one video feed with sufficient quality is already consuming a significant portion of the available bandwidth.

Another problem is the inability to perceive the horizon or attitude as described by Lewis *et al.* [54]. There are many reasons this problem occurs, one of which is the fixed camera. Fixed cameras do not provide feedback on the slope on which the robot is sitting, which often leads to the robot being overturned. Lewis *et al.* [54] proposed a solution that used gravity referenced view instead of fixed view, which they claimed to provide better situation awareness.

When teleoperation is coupled with telepresence, humans are able to interact with remote environments as if they are actually there. However, teleoperation is highly susceptible to communication delays that could potentially result in undesirable outcomes. Unfortunately, the goal of achieving true telepresence (i.e., remote perception is the same as direct perception) is a very difficult problem to solve [46]. Thus, ongoing research is more focused on *functional presence* [46], where there is just enough perception information so that humans can complete their given tasks [110].

The following HRI schemes were developed as a result of the problems encountered with teleoperation, particularly the dependency on nearly instant communications.

Shared Control – the human has control of the robot through high-level instructions.

Typically, the robot is semi-autonomous because the robot is given the autonomy in deciding how to execute the high-level instructions. A typical situation for shared control is waypoint navigation; a set of waypoints is given to the robot and the robot navigates from point to point while avoiding obstacles.

In the shared control scheme, inputs from both the human and robot are required to proceed. The human provides high-level instructions that the robot attempts to achieve. How the robot achieves the high-level instructions is up to the robot. The high-level instructions are transformed into a sequence of primitive commands, which is similar to the direct-control from teleoperation. But the one difference from teleoperation is that the robot is given the freedom to change or modify the sequence of primitive commands based on the robot's own assessment of the environment. Typically, the automation available in robots in this scheme is a form of a reactive automation. The area to which reactive automation is most commonly applied is when humans are responsible for the navigation of remote robots.

One of way that the shared control scheme is used is through *safeguarding* [36, 51]. *Safeguarding* is a way to prevent the robot from being "harmed" by high-level instructions. Whenever a high-level instruction would put the robot in harm's way, safeguarding will override the given high-level instruction to ensure the robot's safety. An example of shared control is the Autonomous Robot Architecture (AuRA) [5]. The AuRA is a hybrid form of *reactive control*, where the robot is "capable of functioning both in the presence or absence of world knowledge, and can reconfigure its [the robot] behaviors based on mission intents, environmental knowledge and success or failure of attaining the mission's goals".

Traded Control – similar to shared control except that the human and the robot are in the same location. The human and robot are required to be in the same location because the process involves the human's mental or physical abilities.

An example of a traded control scheme is the Collision and Accident Avoidance System (CAAS) [43] that helps humans in driving cars safely. The CAAS is supposed to decrease the human driver's mental workload thereby increasing the human driver's situation awareness as well as their comfort level. Goodrich and Boer [43] described two types of automation that are applicable in traded control schemes: *task automation* and *response automation*. *Task automation* occurs when tasks are started by the humans but are easily automated and carried out by the robot so as to help the human (e.g., regulating speed using cruise control). *Response automation* is when tasks are automatically applied by the robot when humans are driving cars (e.g., ensuring safety when changing lanes). According to the Goodrich and Boer [43], the goal of task automation is to "safely promote comfort" while the goal of response automation is to "comfortably promote safety".

Supervisory Control – similar to shared control, the difference is that instead of high-level instructions, humans issue *tasks* [16] to the robot. *Tasks* are similar high-level instructions except that robots have to reason about the tasks (e.g., student A instructs student B "give me the pen" and student B can see two pens but student A only sees one pen). Typically, robots in the supervisory control schemes are mostly autonomous because the robots are able to handle a wide variety of tasks by themselves. This would effectively lead to a lower frequency in the required amount of HRI.

Because humans are issuing tasks, the mental workload on the humans is reduced because the humans are no longer required to form the high-level instructions for the robots. As Adams [1] said, "the underlying . . . technology must be intelligent and achieve complex reasoning that reduces the reliance on the human's cognitive reasoning capabilities". Less reliance on a human's cognitive ability allows more robots to be supervised by a single human. One of the problems in supervisory control schemes is that tasks have to be processed either by the robots or the interface. One issue in understanding the meaning of a task is resolving

ambiguities. For instance, in the example of the two students, how does student B know to which pen student A is referring? The human cognitive processes contextual information such as the line-of-sight of student A, and then infers that "correct" pen is the pen that student A sees. Most of the time, this can be done without student B asking for more information like "which pen?". It would be ideal if such tasks can be issued to robots and the robots can infer their meaning.

One approach to resolving this type of ambiguities is *perspective-taking* [97]. The idea of *perspective-taking* is that the robot would use the resources it has (e.g., cameras and sonars) to model the environment from the perspective of the human that issued the task. In combination to the robot's own perspective of the environment, the robot should then be able to resolve most ambiguities. Trafton *et al.* [97] implemented perspective-taking based on the *polyscheme* cognitive architecture [12]. *Polyscheme* is a cognitive architecture that attempts to create human-level intelligent systems by allowing multiple inference techniques and multiple representations to be integrated in a single system.

Another aspect of research in supervisory control is in how to issue tasks to robots with minimal increase to a human's mental workload without assuming that robots can infer the meaning of tasks. One approach is the use of "tasking" interfaces [65] that use the notion a playbook to issue tasks. According to Miller, Pelican, and Goldman [65], there are three areas that are required for "tasking" interfaces: (1) a shared vocabulary of tasks where humans can issue tasks and a mechanism to know how the tasks are going to be accomplished, (2) enough knowledge from the interface so that intelligent decisions can be made about how to accomplish tasks, and (3) a way to inspect and manipulate the shared vocabulary and an easy and fast way to view the details of the tasks. Parasuraman *et al.* [69] expand on the idea of "tasking" interfaces, where "tasking" interfaces are called delegation-type interfaces. Parasuraman *et al.* [69] proposed a flexible delegation-type interface that increases overall system performance. Parasuraman *et al.* [69] defined flexibility as consisting

of two parts: level of aggregation and level of abstraction. The proposed interface allowed two levels of control: direct control or automated control. However, Parasuraman *et al.* [69] noted in their findings that the benefits of flexibility was negated when the number of robots double from four to eight; probably due to the increase in mental workload when managing more robots.

One of the goals of supervisory control is to allow a single human to supervise multiple robots; the more robots that can be supervised by a single human the better. One aspect of supervisory control is determining what tasks are given to robots. An example can be seen in the work by Adams [1], where she developed an algorithm to control multiple Autonomous Vehicles (AVs) in forming a coalition.

Learning Control – the robot possesses an artificial intelligence that is comparable to human intelligence. HRI is only required because the robot has encountered an unknown situation, where it does not know what to do and requires the expertise and experience of a human expert. Once the robot has learned how to handle the unknown situation, the robot should be able to handle other similar situations that arises.

Unfortunately, this is a future vision in the field of artificial intelligence that has yet to be achieved. Currently, there is no known artificial intelligence system that is equivalent to human intelligence. There are a number of problems to be solved before creating an artificial intelligence that is comparable to human intelligence such as determining when situations or events are equivalent and resolving ambiguities such natural language speech.

Impedance Control – the robots in this control scheme typically are not autonomous, and if the robots do have some autonomy, the autonomy is simple. Generally, robots in the impedance control scheme are used for assisting human actions. For example, a robot could be a prosthetic limb that is attached to a human or an exoskeleton.

Tsuji and Tanaka [99] proposed a mechanism for tracking control properties of the human arm. In addition, they provided a Neural Network (NN) for training their system so that humans can improve their ability to control the robot.

3.5.3 Robot Automation

There are varying degrees of autonomy, from fully autonomous robots to non-autonomous robots. Parasuraman, Sheridan, and Wickens [70] defined ten levels of autonomy. The levels range from non-autonomous (level 1) to fully autonomous (level 10).

1. Robots in this level have no autonomy because humans make all the decisions and perform all the actions for the robots.

2. Robots in this level offer a complete set of decision/action alternatives but the robots play no part in helping humans to choose a decision or action to execute.

3. Robots in this level reduce the set of decision/action alternatives to a few.

4. Robots in this level offer only one decision/action alternative but the decision still lie with the humans.

5. Robots in this level execute the decision/action alternative if approved by humans.

6. Robots in this level give the humans a limited time to "veto" the decision/action alternative before the robots automatically execute the decision/action alternative.

7. Robots in this level automatically execute the decision/action alternative and inform the humans.

8. Robots in this level only inform the humans if asked.

9. Robots in this level decide if the humans should be informed.

10. Robots in this level are fully autonomous because the robots decide everything independent of the humans.

In HRI, when automation in robots increases, the authority given to robots increases as well. Otherwise, humans would be overwhelmed by the task of micromanaging different aspects of multiple robots at the same time, which could potentially lead to a decrease in overall system performance. As the level of automation in robots increases, there is a need for more sophisticated forms of interactions between humans and robots [15]. According to Christoffersen and Woods [15], "the issue is not the level of autonomy or authority, but rather the degree of coordination".

For robots that are not fully autonomous, human interactions are required to maintain a certain level of efficiency or performance. These interactions are an integral part of the system. Crandall *et al.* [18] defined two terms: *neglect time* and *interaction time*. *Neglect time* is "the expected amount of time a robot can be ignored before its performance drops below a threshold" [18] and *interaction time* is "the expected amount of time that a human must interact with a robot to bring it to peak performance" [18]. Crandall *et al.* [18] used the two terms to provide a metric system for determining the validity of different HRIs by evaluating the number of robots in terms of neglect time and interaction time to see if a given HRI is feasible.

Another means of evaluating HRIs is through situation awareness. Scholtz, Antonishek, and Young [83] proposed that supervisory control schemes can be evaluated using situation awareness. Their methodology evaluates supervisory control schemes for each of the three levels of situation awareness [35]: *perception*, *comprehension*, and *projection*. The *perception* evaluation determines if the user interface provides sufficient information to perceive the environment so that tasks can be completed. The *comprehension* evaluation determines how well the user interface combines and interprets information. And lastly, the *projection* evaluation determines how capable the user interface is in predicting the future based on

the current situation.

Most HRIs are designed to meet specific goals. However, that is not necessarily the case. Michaud *et al.* [62] developed a robot named Roball, which did “not have a specific goal to achieve with a clear outcome”. Roball was designed primarily for child development skills. Michaud *et al.* [62] showed that it was possible that HRIs do not have to have specific goals.

3.6 Summary

This chapter covers key background concepts and terms necessary to understand the work in my research. The background includes (1) the OMACS model that is the basis for the CzM model, (2) the GMoDS model that I am extending to allow goal modifications, (3) the O-MaSE methodology that I follow in building the system described in Section 5.2.1, (4) the human factors and mathematical concepts required to capture human factors, and (5) the various aspects to HRIs. The next chapter (Chapter 4) describes the CzM model.

Chapter 4

Chazm Model (CzM)

This chapter presents the CzM that addresses the open question from Chapter 1 about the type of information on humans that should be captured that can lead to better allocation of tasks. Section 4.1 highlights the limitations of the OMACS. Section 4.2 presents the CzM model. Section 4.4 evaluates the CzM model in a humans only scenario and analyzes the time complexity of various task allocation algorithms. Section 4.5 evaluates the CzM in a human-robot scenario. And Section 4.6 summarizes this chapter.

4.1 Limitations of OMACS

OMACS [26] is (1) unable to explicitly capture human performance factors such as *skill*, *training* or *reaction time*, (2) unable to capture information that describe the state of agents (particularly humans) such as *location*, *fatigue*, and *workload*, and (3) unable to capture information pertaining to tasks such as their affect on the state of agents. One use of this information can be for better task allocation. By definition, human performance factors are factors that play a part in the performance of an individual human with respect to some specific task. By including a human performance factor that contributes to the performance,

the computation of the performance of an individual becomes more accurate.

In OMACS, the *possesses* function is defined as $possesses : Agent \times Capability \mapsto [0, 1]$, which indicates how well an agent can use the capability. It has been suggested that all information can be captured as capabilities and the scores used to reflect the actual values. However, that is generally not the case. Silverman [84] defined two types of information: quality-type and quantity-type. Quality-type information has a range $[0, 1]$ while quantity-type information is unbounded. The range $[0, 1]$ is indicative of quality-type information. However, there is also quantity-type information that must be captured. Some quantity-type information can be transformed into quality-type information without loss of meaning such as *fatigue*. *Fatigue* can be represented as quality-type information because there is a known minimum and maximum level. Unfortunately, not all quantity-type information have known minimum and maximum values. An example of a quantity-type information that cannot be transformed to quality-type without loss of meaning is *location*. For example, how does the longitude and latitude values convert to a value between $[0, 1]$ without loss of meaning?

Another limitation of the *possesses* function is the predefined meaning associated with the range $[0, 1]$, where 0.0 means a lack of or a broken capability and 1.0 means complete or perfect capability. In other words, higher values are better. However, there are some quality-type information that are more intuitive if the reverse is true such as *fatigue*; for *fatigue*, lower values are better.

In OMACS, the definitions of the *achieves* function and the `rcf` function do not allow information about goals to be used. The *achieves* function is defined as $achieves : Goal \times Role \mapsto [0, 1]$, which indicates how well a role achieves a goal. However, the `rcf` function does not account for variations of the same goal type. For example, for the tasks *survey A1* and *survey A2* (from Chapter 2) have the same role and the same goal type; the only difference is in the parameter values of the goal instances. Thus, for two equally capable

agents, preference may be given to the agent that is physically closer to the area. Similarly, the `rcf` function is defined as $rcf: Agent \times Role \mapsto [0,1]$, which determines how well an agent performs a role. In order to determine how well an agent can perform a role to achieve a goal, an overall score is computed by combining score from the `rcf` function with the score from the *achieves* function. However, this does not allow parameters values of goals to affect the overall score such as giving preference to agents that are physically closer to the area.

The next section (Section 4.2) introduces a model that addresses the above limitations.

4.2 Chazm Model (CzM)

This section introduces the CzM that captures human performance factors that can be used by task allocation algorithms. A number of additions and changes to OMACS are required to capture human performance factors so that human performance factors can be used in task allocation algorithms; these additions and changes result in the creation of the CzM model. Figure 4.1 shows some of the additions and changes to OMACS.

There are three types of elements in Figure 4.1: rectangles, lines, and ellipses. Rectangles are entities. Lines between entities are relations; the arrows on the lines denote direction for reading purposes only. For instance, role X *requires* capability Y. Ellipses attached to relation via a dashed line indicate values associated with relations that are functions. Elements that are greyed out are elements that exists in OMACS. In CzM, there are four new entities, six new relations (two of which are functions), and changes to existing elements.

The first new entity is an *attribute*. An *attribute* describes a property of an agent. Currently, there are three types of attributes: quality-type, quantity-type, and unbounded-type. A quality-type attribute constrains the value to the $[0,1]$ range, a quantity-type attribute constrains the range to $[0,+\infty]$, and an unbounded-type attribute does not constrain the values $[-\infty,+\infty]$. In addition, each type is either a positive-type or negative-

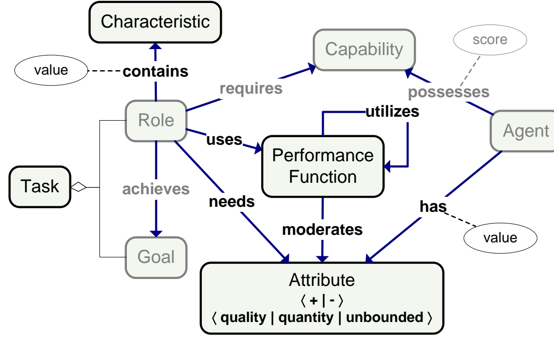


Figure 4.1: CzM Model

type attribute, which indicates the type of scale used to measure the values in relation to one another. Some attributes can be represented as either a positive-type (the higher the value, the better) or a negative-type (the lower the value, the better). For example, consider the attributes *energy* and *fatigue*. These two attributes represent the same concept except that for energy, higher values are better, while for fatigue, lower values are better.

The purpose of the second new entity, *performance function*, is to capture the PMFs. Capturing PMFs as an entity allows user-defined PMFs to be used at runtime. For instance, two roles may have slightly different PMFs for computing the *fatigue* of agents after performing different roles because one role may require more strenuous activities than the other. PMFs are captured in CzM as functions of the form of Definition 4.1. The *Role*, *Agent*, and *Goal* inputs inform the PMF function to which role the agent is performing to achieve the goal. The $Set\{Assignment\}$ is the relevant set of assignments for the PMF function, which can be all the assignments of the organization or a subset such as the assignments of a particular agent; not all assignments affect the computation of PMFs.

$$\text{pmf}_{\text{attribute}} : \text{Role} \times \text{Agent} \times \text{Goal} \times \text{Set}\{\text{Assignment}\} \mapsto \text{value} \tag{4.1}$$

The *characteristic* entity describes a property of a role. A *characteristic* provides additional information that can be utilized by *performance functions*. For example, the

surveyor role (from Chapter 2) may contain information about the average length of time taken to complete the role, which can be captured as the *average completion time* characteristic. The *average completion time* characteristic can be used by *performance functions* associated with the *surveyor* role.

The last new entity is a *task*. A *task* is the composition of a role and a goal. The purpose of the *task* entity is purely for human understanding; computationally, a *task* does not provide any additional information other than what the associated role and goal already provide. For example, the task *survey A1* (from Chapter 2) is comprised of the role *surveyor* and the goal *survey A1*. In OMACS, an assignment is formally defined as $assignment : Agent \times Role \times Goal$. In CzM, the definition of an assignment is expanded to include $assignment : Agent \times Task$.

The *has* function (Definition 4.2) takes in an agent and an *attribute* and returns a value consistent with the type of that attribute: quantity $[0, +\infty]$, quality $[0, 1]$, or unbounded $[-\infty, +\infty]$. Even though the *has* function specifies a relation between an agent's *attribute* and a single value, it is straightforward to model complex attributes such as compound attributes. A compound attribute such as *location* does not contain a value but is comprised of multiple single values. For example, the *location* attribute is typically comprised of three values: longitude, latitude, and altitude. The three values can be represented as three attributes: *longitude*, *latitude*, and *altitude*. A logical grouping of the three attributes (*longitude*, *latitude*, and *altitude*) into the *location* attribute would not provide any functional benefits. To ease the use of CzM, design tools can provide logical groupings for complex attributes such as *location*. These design tools would then translate these complex attributes for use in CzM.

$$\text{has} : \text{Agent} \times \text{Attribute} \mapsto \text{value} \tag{4.2}$$

The *moderates* relation (Definition 4.3) specifies a relation between a *performance function* and an *attribute*. Because a *performance function* captures a PMF and a PMF computes the result for a particular *attribute*, the *moderates* relation is a many-to-one relation (i.e., a *performance function* moderates exactly one *attribute* but an *attribute* can be moderated by multiple *performance functions*). The *moderates* relation specifies the *attribute* to which the result of the PMF is applicable. For example, to capture a PMF that computes fatigue, the PMF is captured as a *performance function* that *moderates* the *fatigue* attribute.

$$\text{moderates} : \text{Performance Function} \times \text{Attribute} \tag{4.3}$$

The *needs* relation (Definition 4.4) specifies a relation between a role and an *attribute*. The purpose of the *needs* relation is to capture additional requirements for performing a role beyond just capabilities as currently used in OMACS. The *needs* and *requires* relations specify the complete set of requirements an agent must meet to perform a role. For example, to be assigned to the task *survey A1* (from Chapter 2), the role *surveyor* requires the *training* capability and the *rank* attribute and the goal *survey A1* contains the exact requirements; an agent can be of any rank and must have basic training.

$$\text{needs} : \text{Role} \times \text{Attribute} \tag{4.4}$$

The *uses* relation (Definition 4.5) specifies a relation between a role and a *performance function*. The purpose of the *uses* relation is to indicate which of the *attributes* associated with the role through the *needs* relation require the use of a PMF to compute the value. For example, the *reaction time* attribute may not need a PMF because the value is obtained directly from the agent through the *has* function. But the *fatigue* attribute may need a PMF to compute the value because the result may depend on the roles (e.g., *surveyor* role,

identifier role, *rescuer* role). More importantly, the *uses* relation differentiates between attributes whose values are used and attributes whose values are changed as a result of performing roles. For correctness, there are two constraints on the *uses* relation: (1) a role can only use a performance function if the attribute modified by the performance function is also the attribute needed by the role (Constraint 4.6) and (2) a role cannot use two or more performance functions that moderate the same attribute (Constraint 4.7).

$$\text{uses} : \text{Role} \times \text{Performance Function} \tag{4.5}$$

$$\begin{aligned} &\forall r \in \text{Role}, f \in \text{Performance Function}, a \in \text{Attribute} \\ &\quad \mid (r, f) \in \text{uses} \land (f, a) \in \text{moderates} \Rightarrow (r, a) \in \text{needs} \end{aligned} \tag{4.6}$$

$$\begin{aligned} &\forall r \in \text{Role}, f, f' \in \text{Performance Function}, a \in \text{Attribute} \\ &\quad \mid (r, f), (r, f') \in \text{uses} \land (f, a), (f', a) \in \text{moderates} \Rightarrow f = f' \end{aligned} \tag{4.7}$$

The *utilizes* relation (Definition 4.8) specifies a relation between two *performance functions*. The reason for the *utilizes* relation is to indicate whether a *performance function* uses another *performance function* for computation. For example, to compute the overall workload, the overall workload PMF may require the auditory workload PMF, cognitive workload PMF, and visual workload PMF. There are two constraints on the *utilizes* relation: (1) the *utilizes* relation do not form a cycle (Constraint 4.9) and (2) if a role uses a *performance function* (A), which utilizes another *performance function* (B), then the *attribute* moderated by *performance function* (B) is also needed by the role (Constraint 4.10). The transitive closure of a relation is denoted by the $^+$ symbol.

$$\text{utilizes} : \text{Performance Function} \times \text{Performance Function} \tag{4.8}$$

$$\forall f \in \text{Performance Function} \mid (f, f) \notin \text{utilizes}^{+} \tag{4.9}$$

$$\begin{aligned} &\forall r \in \text{Role}, f, f' \in \text{Performance Function}, a \in \text{Attribute} \\ &\quad \mid (r, f) \in \text{uses} \wedge (f, f') \in \text{utilizes}^{+} \wedge (f', a) \in \text{moderates} \\ &\quad \Rightarrow (r, a) \in \text{needs} \end{aligned} \tag{4.10}$$

The *contains* function (Definition 4.11) takes in a role and a *characteristic* and returns a value. For example, surveying an area would take on average 30 minutes. This example can be modeled as the *surveyor* role *contains* the *average completion time* characteristic with a value of 30. Then a *performance function* for computing the *fatigue* for that role can use the *average completion time* characteristic for computing the new *fatigue* value of agents after performing the *surveyor* role.

$$\text{contains} : \text{Role} \times \text{Characteristic} \mapsto \text{value} \tag{4.11}$$

In OMACS, to perform a role, an agent must have the required capabilities. In CzM, to perform a role, an agent must have the required capabilities and the necessary attributes. The `rcf` function defined in OMACS is defined as $rcf : Role \times Agent \mapsto [0, 1]$ and the `rcf` function only evaluates the capabilities of an agent with respect to the role. This definition is no longer sufficient due to the addition of attributes; thus, the `rcf` function is not part of CzM. Instead, a `goodness` function (Definition 4.12) is defined that evaluates both the capabilities and attributes required by agents. Furthermore, the specific goal being pursued is also part of the input for the `goodness` function because the goal may contain parameters that affect how well agents may perform a particular role. The $Set\{Assignment\}$ is the relevant set of assignments for the `goodness` function, which can be all the assignments of the organization or a subset such as the assignments of a particular agent as not all

assignments affect the computation of PMFs. The `goodness` function has one constraint (Constraint 4.13), where the return value must be 0.0 if the agent do not possesses a required capability, a needed attribute, or the role cannot achieve the goal.

$$\text{goodness}_{\text{role}} : \text{Role} \times \text{Agent} \times \text{Goal} \times \text{Set}\{\text{Assignment}\} \mapsto [0,1] \tag{4.12}$$

$$\text{goodness}(r,a,g,\phi) = \begin{cases} 0.0 & \text{if } \exists c \in (r,c) \in \text{requires} \mid (a,c) \notin \text{possesses} \\ & \quad \vee \exists n \in (r,n) \in \text{needs} \mid (a,n) \notin \text{has} \\ & \quad \vee (r,g) \notin \text{achieves} \\ [0,1] & \end{cases} \tag{4.13}$$

Due to the removal of the `rcf` function from CzM, the *capable* function is eliminated. In OMACS, the *capable* function is defined as $\textit{capable} : \textit{Agent} \times \textit{Role} \mapsto [0,1]$ with the constraint that return value is the same as the `rcf` function: $\textit{capable}(a,r) = \textit{rcf}(a,r)$. Thus, the *capable* function is redundant. The existing definition of the *capable* function from OMACS is insufficient to correctly capture the `goodness` function because even if $\textit{capable}(a,r) > 0$, it is still possible for $\textit{goodness}(a,r,g,\phi) = 0.0$. A different solution would be to redefine the *capable* function as $\textit{capable}(a,r,g,\phi)$ but then that would mean that $\textit{capable}(a,r,g,\phi) = \textit{goodness}(a,r,g,\phi)$ and therefore the *capable* function is redundant.

In OMACS, the *achieves* function was defined as $\textit{achieves} : \textit{Role} \times \textit{Goal} \mapsto [0,1]$, which indicates how well the role achieves a specific goal. However, in CzM, *achieves* is defined as a simple relation between a role and a goal (Definition 4.14). The reason for the change is because the `goodness` function takes in a goal as input, the score functionality is now part of the `goodness` function. This change provides greater flexibility because CzM allows specific goals to contribute to the `goodness` score. For example, there are two agents capable of

surveying $A1$ and sometimes it is preferable to select the agent that is closer to $A1$.

$$\text{achieves} : \text{Role} \times \text{Goal} \tag{4.14}$$

There is a brief discussion in Appendix C that describes a general guideline to creating runtime models.

4.3 Evaluation Process

The next two sections (Section 4.4 and Section 4.5) evaluate the CzM model for the purpose of task allocations. However, PMFs research is currently in the early stages. Many studies in the past have shown a causal relation between PMFs and human performance. However, the relation has not been properly defined for general use across multiple domains. For example, fatigue as a function of lack of sleep has been extensively studied by the U.S. Department of Transportation to specify regulations for businesses for hours-of-service[1] of drivers. However, it is not clear how the results from the studies translate to, for example, construction workers of high-rise buildings as it is unknown whether working continuously in construction for x and $x + 1$ hours correspond to a similar increase in the risk of causing an accident as driving continuously for x and $x + 1$ hours.

For PMFs to be useful for task allocation purposes, the relation between PMFs and performance needs to be quantified. For instance, in order to properly use PMFs, there is a need for detailed knowledge such as if fatigue increases by x amount, performance should decrease by y amount. PMFs research is not at this stage yet. Thus, the PMFs in the evaluation scenarios do not represent validated PMFs for the associated domains.

[1] More information can be obtained from http://www.fmcsa.dot.gov/rules-regulations/topics/hos/index.htm.

4.4 Multiple Humans Evaluation

The Conference Management System (CMS) [22, 111] is used as a basis for evaluating the usefulness of CzM versus OMACS. The CMS represents a conceptual model of the process that takes place leading up to a scientific conference, where authors submit their papers, reviewers are given papers to review, the PC chair makes decisions to accept papers, and accepted papers are sent to the printers for printing. Figure 4.2 shows the GMoDS model that captures the CMS process, where system goals are represented and further decomposed into subgoals. The top-level goal of the CMS is to *manage submissions*, which is decomposed into six conjunctive subgoals, which are also further decomposed into subgoals. At the bottom of the goal tree are the leaf goals, which are *collect papers*, *distribute papers*, *partition papers*, *review paper*, *collect reviews*, *make decision*, *inform declined*, *inform accepted*, *collect finals*, *master CD*, and *print proceedings*. These leaf goals are the only goals that can be pursued by agents to achieve the top-level goal.

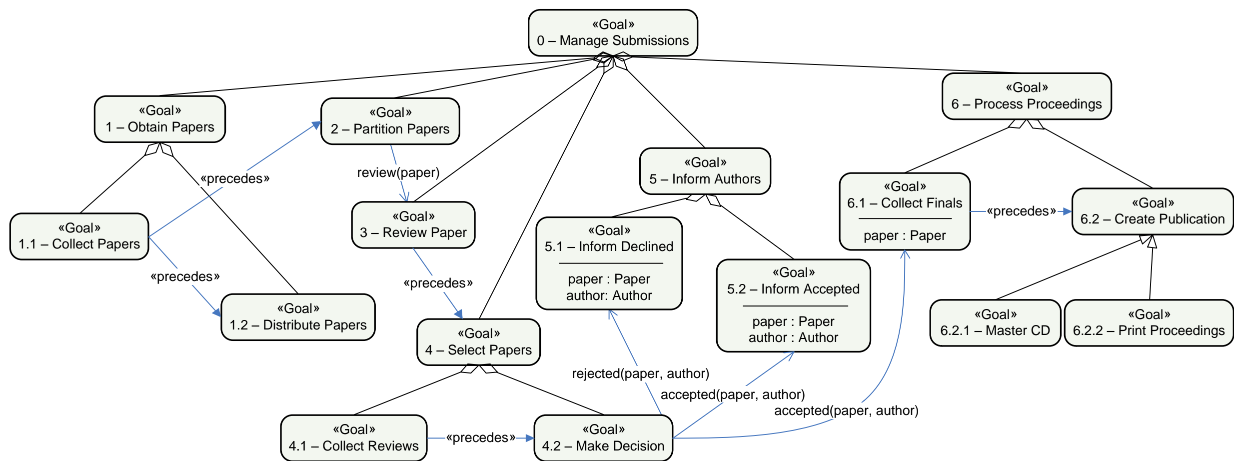


Figure 4.2: CMS Goal Model

Figure 4.3 shows the role model where each leaf goal is mapped to a specific role. These roles are defined to achieve their associated leaf goal(s) and specify the capabilities required by agents in order to perform them. For the experiments, a minor modification to the

role model is required to include the necessary attributes. The role *reviewer* is modified to include *workload*, *stress*, and *incentive* as necessary attributes.

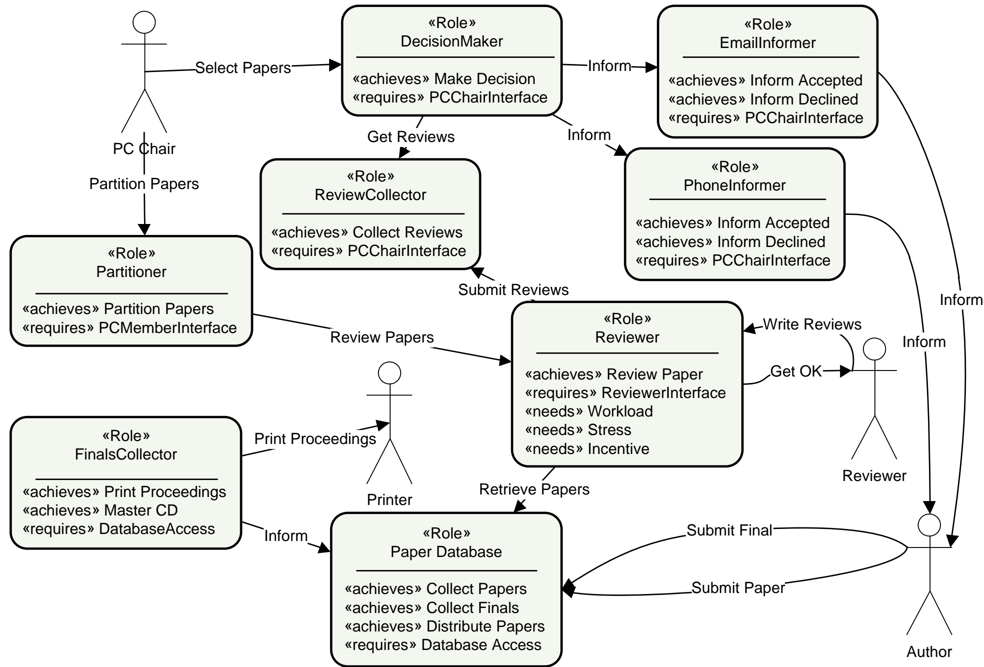


Figure 4.3: CMS Role Model

The workload PMF, which is the sum of the workload values of each paper the agent is reviewing, is defined by two equations: Equation 4.15 and Equation 4.16. In Equation 4.15, *p.type* refers to the type of paper. In Equation 4.16, *g.paper* refers to the paper parameter of the goal g.

$$\text{workload}(p) = \begin{cases} 10 & \text{if } p.\text{type} = \text{poster} \\ 20 & \text{if } p.\text{type} = \text{short} \\ 40 & \text{if } p.\text{type} = \text{full} \end{cases} \tag{4.15}$$

$$\text{pmf}_{\text{workload}}(r, a, g, \phi) = \left(\sum_{(a,r,g') \in \phi} \text{workload}(g'.\text{paper}) \right) + \text{workload}(g.\text{paper}) \tag{4.16}$$

4.4.1 Experimental Setup

An experiment evaluates the usefulness of CzM by using different task allocation algorithms with a given number of reviewers, a given number of papers to review, a given number of papers to accept, and a given range for the quality of a submitted paper. The CMS experiments are set up such that for every experiment, the number of reviewers is fixed at 50, the number of papers accepted is fixed at 40, and the submitted paper quality range is from $[45\%, 55\%]$. The range is kept small so as to increase the chance of a paper that is not in the top 40 being accepted due to inaccurate reviews of that paper; the small interval makes the problem harder because it is harder to discriminate between papers. Each submitted paper is given a quality that is randomly selected from the given range $[45\%, 55\%]$. These submitted papers are ranked based on their quality; ideally, only the top 40 papers are accepted. There are a total of 80 experiments. The first experiment starts at 40 papers to review, the second at 41 papers to review, the third at 42 papers to review, and so forth, up to the 80th experiment with 120 papers to review. Each submitted paper is reviewed by three reviewers. Once all reviews are in, the decision to accept or reject a paper is based on the three reviews. The purpose of an experiment is to evaluate how well a task allocation algorithm can assign these reviews to reviewers so that the set of accepted papers is as close as possible to the ideal set.

In each experiment, the performance of four reorganization algorithms is compared. Three of the algorithms are general reorganization algorithms, which means that the algorithms use only information directly available in the model, while the fourth algorithm is not entirely general. The implementation of the fourth algorithm utilizes information about the domain. Although the implementation of the fourth algorithm is not general, it is possible to generalize the implementation such that it is applicable to other domains. However, generalizing the implementation is beyond the scope of this dissertation. The three general reorganization algorithms are random, round robin, and attributes-greedy; the

non-general algorithm is the attributes-enhanced. Since the OMACS model only provide information on capabilities and, in this experiment, all reviewers are equally capable, randomly assigning papers or assigning papers in a round robin is all OMACS can do. On the other hand, the CzM model provides additional information about reviewers which is used by the attributes-greedy and attributes-enhanced algorithms. These algorithms are discussed in detail in Section 4.4.2. Although the goal model captures the entire CMS process, the focus of the experiments is on allocating the instances of the *review paper* goal.

There are three types of reviewers defined: tenured professors, assistant professors, and graduate students. All reviewers need three attributes (as shown in Figure 4.3): *incentive*, *stress*, and *workload*. These three attributes are of the same scale, where a value of 0 means no incentive, no stress, and no workload respectively[2]. *Incentive* values are none, low, medium, and high, maximum. For computational purposes, the incentive values are mapped to 0, 30, 50, 70, and 100 respectively. *Stress* and *workload* are measured in terms of percentages and do not have an upper-bound. These attributes determine the maximum number of papers a reviewer can review before becoming overloaded/overburdened. An overloaded reviewer will produce reviews that are less than 100% quality. As *incentive* increases, a reviewer is able to review more papers before becoming overburdened. As *stress* decreases, a reviewer is able to review more papers before becoming overburdened. Similarly, as *workload* decreases, a reviewer is able to review more papers before becoming overburdened. Table 4.1 shows the starting values of the attributes for the three types of reviewers.

	Incentive	Stress	Workload
Tenured Professors	low (30)	0%	0%
Assistant Professors	medium (50)	50%	0%
Graduate Students	low (30)	60%	0%

Table 4.1: Attribute Values of Agent Types

[2]Determining what the values means in terms of numbers is beyond the scope of this dissertation.

There are three types of papers defined: full paper, short paper, and poster paper. Reviewing a full paper would add 40% to a reviewer's workload; reviewing a short paper adds 20% to the workload; and reviewing a poster paper adds 10% to the workload. The quality (q_r) of a review produced by a reviewer is defined by Equation 4.17. For example, if a tenured professor has 6 short papers to review, the *workload* PMF will return a result of 120% workload, which results in the quality of all 6 reviews being $100 \div (120+30-0) \times 100 = 66.\overline{6}\%$.

$$\begin{aligned} &\text{workload} = \text{pmf}(\text{Role}, \text{Agent}, \text{Goal}) \\ &\text{total load} = \text{workload} + \text{stress} - \text{incentive} \\ &q_r = \begin{cases} 100 & \text{if total load} < 100, \\ \dfrac{100}{\text{total load}} \times 100 \end{cases} \end{aligned} \tag{4.17}$$

Incentive and stress do not change throughout the experiment. Workload is computed based on the number of papers given to a reviewer, with each paper contributing either 10%, 20%, or 40% to the reviewer's workload. The quality of a review (q_r) and the quality of the paper (q_p) determines the review score (s) as defined in Equation 4.18. As the review quality (q_r) approaches to 0, the range of possible review scores approaches $[0, 100]$. For example, if $q_p = 60$ and $q_r = 80$, then $s = 60 + [-10, 10] = [50, 70]$.

$$s = \begin{cases} q_p & \text{if } q_r = 100, \\ q_p + \left[-\dfrac{100 - q_r}{2}, \dfrac{100 - q_r}{2}\right] \end{cases} \tag{4.18}$$

Once all reviews are done, the average review score is computed for each paper since there are three reviews per paper. Once the average review score is computed, papers are sorted by the average review score and the top 40 are accepted.

The distribution of the reviewers (n) remain the same for each experiment. There are approximately $n/3$ for each type of reviewers. The mathematical distribution for each type

are as follows: tenured professors (r_{tp}) are $\lfloor \frac{n}{3} \rfloor$, assistant professors ($r_{ap}$) are $\lfloor \frac{n}{3} \rfloor$, and graduate students (r_{gs}) are $n - r_{tp} - r_{ap}$. Since there are 50 reviewers in the experiments, there are 16 tenured professors, 16 assistant professors, and 18 graduate students.

Because of the randomness in various aspects of the experiments such as the random paper qualities and the bounded-random error for review scores, each experiment is executed $10,000$ times to normalize the data.

4.4.2 Algorithms

The *random* algorithm randomly selects an agent capable of achieving a goal and assigns that goal to the agent. This process continues until all goals have been assigned. Because the random algorithm only cares about finding an agent that is capable of achieving a given goal, the random algorithm only uses the score of the `goodness` function to check that the agent is capable (i.e., the `goodness` function score is greater than 0.0). The `goodness` function is defined by Equation 4.19, which is the same as Equation 3.1 in Chapter 3, for all roles. For the purpose of this dissertation, the results from the random algorithm act as the baseline for the other algorithms.

$$\sqrt[|\{c|(r,c)\in\text{requires}\}|]{\prod_{c\in\{c|(r,c)\in\text{requires}\}} \text{possesses}(a,c)} \tag{4.19}$$

The *round robin* algorithm assigns goals by evenly distributing the goals to capable agents. The `goodness` function for all roles is defined by Equation 4.19. Because not all agents are capable of achieving all the goals, the round robin algorithm keeps track of the number of goals assigned to each agent. For a given goal, the capable agent (`goodness` > 0.0) with the least number of goals currently assigned is assigned to that goal. This process continues until all goals have been assigned. Because the order of the goals impact the outcome (particularly the *review paper* goals, because a reviewer can only review a

paper once), the goals are sorted to keep the *review paper* goals of the same paper together. Furthermore, the ordering of agents also affects the outcome; it could result in better or worse assignments. For example, if there are 3 reviewers (R_1 and R_2), and they can review a maximum of 1, 2 papers respectively before being overburdened, and there are 3 papers to review. If the order is R_1 and R_2, then R_1 will get 2 papers, which overburdens R_1. However, if the order is R_2 and R_1, then no agent is overburdened. Since the experiments are executed $10,000$ times, the ordering of agents is randomized to normalize the results.

The *attributes-greedy* algorithm uses the `goodness` function score to rank all agents for a given goal and assigns the agent with the highest score to that goal. Because CzM allows access to the workload, stress, and incentive values of an agent, the `goodness` function for the *reviewer* role is defined as computing the review quality (q_r). So the `goodness` function returns the value of q_r as defined by Equation 4.17. For the rest of the roles, the `goodness` function is defined by Equation 4.19. This process continues until all goal are assigned.

Similarly, for the *attributes-enhanced* algorithm, the `goodness` function for the *reviewer* role computes q_r as defined by Equation 4.17. In addition, the attributes-enhanced algorithm tracks the contributions (Δ) of an agent as shown in Equation 4.20[3] and uses it to determine the best agent instead of just relying on the basic `goodness` score. The `goodness` function only captures the score of an agent performing the role in the current context (i.e., all the assignments of the agent). But the `goodness` function does not recomputes the scores of existing assignments, which may change if a new goal is assigned to the agent. For example, an agent is assigned one paper to review and the `goodness` function score is 1.0, which means that the agent will produce a review with 100% quality. If that agent were to be assigned a second paper to review and the `goodness` function score is 0.9, that means that the agent would produce two reviews with 90% quality. The attributes-enhanced algorithm does this "recomputation" of existing assignments by using Equation 4.20, which tracks

[3]As mentioned earlier, the equation is a result of analyzing the domain but the equation can be generalized.

each agent's overall contribution by comparing the current contribution (two 90% quality reviews) with the previous contribution (one 100% quality review).

$$\Delta_i = \texttt{goodness} * (\text{assignments} + 1) - \Delta_{i-1} \tag{4.20}$$

4.4.3 Attributes Only Results

There are three types of data collected in the experiments: *score difference*, *set commonality*, and *review quality*. *Score difference* measures the sum of the accepted paper qualities versus the sum of the ideal paper qualities. *Set commonality* measures the percentage of ideal papers in the set of accepted papers. *Review quality* measures the average review quality of all reviews. In the experiments, there are two factors that significantly impact the performance of algorithms: the number of assignments[4] for each agent and the quality of reviews produced by each agent. The quality of reviews factor can only be measured accurately by algorithms that have access to the workload, stress, and incentive attributes because these attributes affect the quality of a review as defined by Equation 4.17.

There is a relationship between the two factors; as the number of assignments increases, the quality of reviews tend to decrease. However, the importance of the two factors are not constant throughout the experiments. Because the number of reviewers are fixed at 50 for all experiments, the number of assignments is less important than quality of reviews when the number of submitted papers are low. However, as the number of submitted papers increases, the importance of the number of assignments also increases to a point where the number of assignments becomes more important than quality of reviews. Also, the importance of the two factors depends the measurement system. For example, the quality of reviews factor plays a more important part in the *review quality* measurement than the other two measurements. Based on the relationship between the two factors, the hypothesis

[4]In these experiments, the number of assignments is the same as the number of papers to review.

is that the results are generally split into three parts: (1) the first part is where the quality of reviews factor is the dominant factor while the number of assignments factor is minor, (2) the second part is where the two factors are equally important, and (3) the third part is where the number of assignments factors is the dominant factor while the quality of reviews factor is minor.

The performance of the algorithms are linked to how the algorithms use the two factors. Although the random algorithm ignores both factors, indirectly and to a certain extent through random selection, the random algorithm utilizes the number of assignments factor. The round robin algorithm considers only the number of assignments factor and ignores the quality of reviews factor. The attributes-greedy algorithm considers only the quality of reviews factor and ignores the number of assignments factor. The attributes-enhanced algorithm considers both factors.

Figure 4.4 shows the results of the four algorithms as measured by the *score difference*. There are three points of interest in the graph. The first point of interest (around 52 submitted papers) is where the round robin algorithm begins to accept papers that are not in the top 40 papers because some reviewers are overburdened (i.e., producing reviews that are less than 100% quality) from being assigned too many papers to review. After this point, it is still possible to keep all reviewers from being overburdened. Both the attributes-greedy and attributes-enhanced algorithms that use CzM are able to produce assignments that keep the set of accepted papers the same as the top 40 papers. The second point of interest (around 72 submitted papers) is where the effect of overburdened reviewers becomes noticeable for the attributes-greedy and attributes-enhanced algorithms. After this point, the attributes-greedy and attributes-enhanced algorithms still produce better results than the random and round robin algorithms. At the third point of interest (around the 98 submitted papers), the performance of the attributes-greedy and round robin algorithms converge. This is the point where the number of assignments factor is just as important as

the quality of reviews factor. The attributes-enhanced algorithm maintains a performance advantage over the other algorithm.

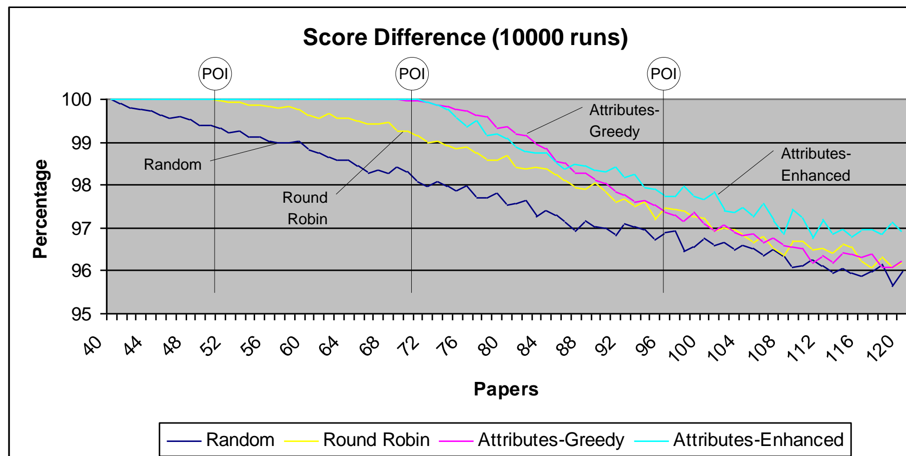


Figure 4.4: Score Difference Graph

Figure 4.5 shows the results of the four algorithms as measured by *set commonality*. There are three points of interests in the graph. At the first point of interest (around 52 submitted papers), the round robin algorithm begins to fall off while the attributes-greedy and attributes-enhanced algorithms still produce assignments that keep all reviewers from being overburdened because there are 50 reviewers and about 52 papers to be reviewed three times. The second point of interest (around 70 submitted papers) is where the effect of overburdened reviewers becomes noticeable for the attributes-greedy and attributes-enhanced algorithms. Still, the attributes-greedy and attributes-enhanced algorithms maintain an advantage over the random and round robin algorithms. At the third point interest (around 90 submitted papers), the performance of the attributes-greedy algorithm converges to the performance of the round robin algorithm but the attributes-enhanced algorithm still maintains an advantage over the other algorithms. The performance advantage of the attributes-greedy algorithm over the round robin algorithm is gone because, around the third point of interest, the number of assignments factor is just as important as quality of reviews factor. However, the attributes-enhanced algorithm still maintains an

advantage over the other algorithms because it uses both factors (number of assignments and quality of reviews) in the form of overall contributions (Equation 4.20).

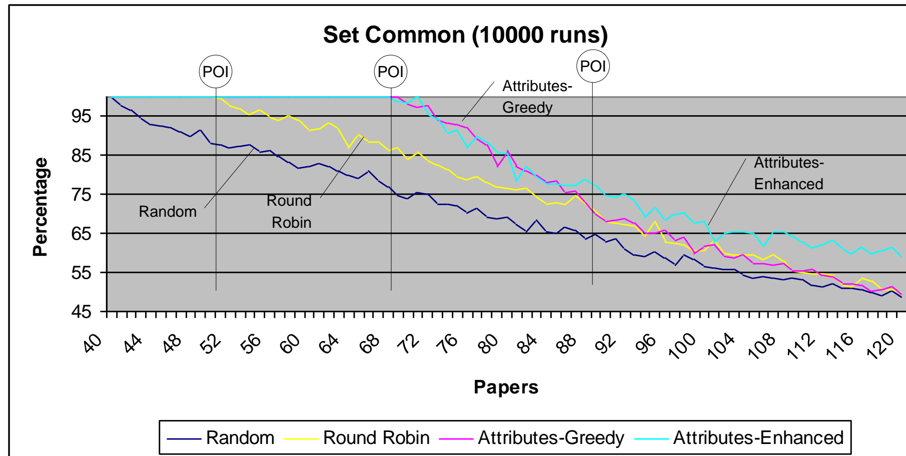


Figure 4.5: Set Commonality Graph

Figure 4.6 shows the results of the four algorithm as measured by *review quality*. Again, there are three points of interests in the graph. The first point of interest (around 52 submitted papers) is where the round robin algorithm begins to produce assignments that result in reviews that are less than 100% quality. The attributes-greedy and attributes-enhanced algorithms still produce assignments that result in 100% quality reviews. The second point of interest (around 72 submitted papers) is where the effect of overburdened reviewers becomes noticeable for the attributes-greedy and attributes-enhanced algorithms. Up to the third point of interest (around 88 submitted papers), the attributes-greedy and attributes-enhanced algorithms maintain a noticeable advantage over the random and round robin algorithms. However, after this point, the attributes-greedy begins to perform worse than the round robin algorithm, while the performance advantage of the attributes-enhanced algorithm over the round robin algorithm becomes smaller. Again, the attributes-enhanced algorithm maintains an advantage over the other algorithms because it uses both factors (number of assignments and quality of reviews) in the form of overall contributions (Equation 4.20). Because the attributes-greedy algorithm only considers review quality and

the number of assignments becomes more important than quality of reviews, the attributes-greedy algorithm begins to perform worse than the round robin algorithm.

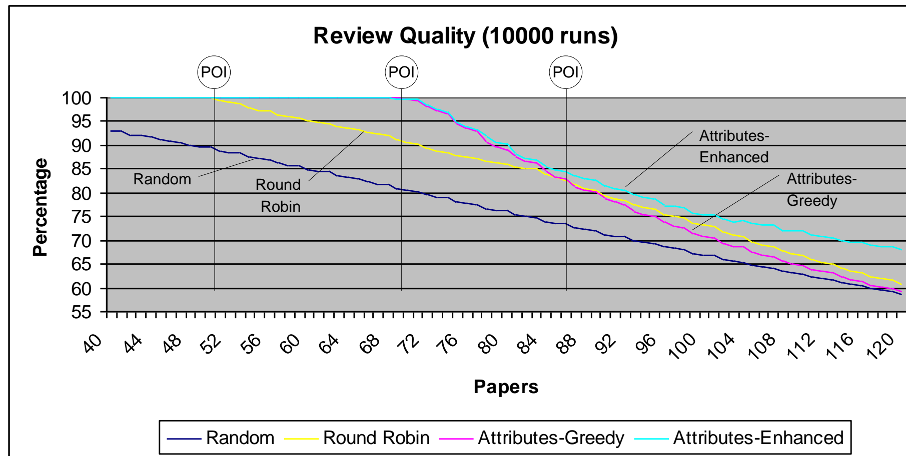


Figure 4.6: Review Quality Graph

Based on the three graphs, the attributes-enhanced algorithm is able to produce better assignments due to the use of the two factors: number of assignments and quality of reviews. On the other hand, the attributes-greedy algorithm, which uses only the quality of reviews factor, is only able to maintain an advantage over the round robin algorithm for the early portions of the graph; up to the point when the number of assignments factor becomes dominant factor. The performance of the attributes-greedy algorithm either converges to the performance of the round robin algorithm or performs worse than the round robin algorithm because the number of assignments factor, which is ignored by the attributes-greedy algorithm, becomes just as important or more important than the quality of reviews factor. This leads to the conclusion that just having the information available is not sufficient. The information needs to be used in the proper context, which leads to the next section that explores the case where the scores of capabilities matter.

4.4.4 Attributes and Capabilities Results

The setup of CMS experiments in Section 4.4.1 assumes that every agent (in this case, the reviewers) are equally capable in their reviewing abilities. However, this is not generally the case. The setup of the experiments in this section uses different scores for the review capability to reflect the ability of the three reviewer types. Table 4.2 shows the starting values of the attributes and the reviewing capability of the three types of reviewers, where a value of 1.0 means 100%. The reviewing capability of the reviewers do not change throughout the experiments.

	Incentive	Stress	Workload	Review Ability
Tenured Professors	low (30)	0%	0%	1.0 (100%)
Assistant Professors	medium (50)	50%	0%	0.8 (80%)
Graduate Students	low (30)	60%	0%	0.6 (60%)

Table 4.2: Attribute and Capability Values of Agent Types

Since the capability scores are now different, a greedy algorithm would not return the same assignments as the round robin algorithm as the experimental setup in Section 4.4.1. The `goodness` function for the *reviewer* role for the greedy and round robin algorithms is defined by Equation 4.19. The greedy algorithm makes assignment decisions based on Equation 4.21 for all agents. If the numerator is always the same value, then the greedy algorithm is equivalent to the round robin algorithm when making assignments.

$$\frac{\text{goodness}(r, a, g)}{\text{number of papers for } a} \tag{4.21}$$

In order to incorporate capability scores in the experiments, a minor change to Equation 4.17 is required. Equation 4.22 defines how review quality is computed that uses the score of an agent's reviewing capability. In Equation 4.17, max load is always 100. In Equation 4.22, max load is multiplied by the score of the agent's reviewing capability, which ranges $[0, 1]$. And thus, tenured professors have 100 max load, assistant professors

have 80 max load, and graduate students have 60 max load.

$$
\begin{aligned}
&\text{max load} = 100 \times \text{reviewing capability} \\
&\text{total load} = \text{workload} + \text{stress} - \text{incentive} \\
&q_r = \begin{cases} 100 & \text{if total load} < \text{max load}, \\ \dfrac{\text{max load}}{\text{total load}} \times 100 \end{cases}
\end{aligned}
\tag{4.22}
$$

Similar to Section 4.4.3, there are two factors to consider in the experiments: number of assignments and quality of reviews. The greedy algorithm considers the number of assignments factor and, in a limited degree, considers the quality of reviews by using just the capability score while ignoring the attributes. Likewise, the `goodness` function for the attributes-greedy and attributes-enhanced algorithms were changed to match Equation 4.22. All other aspects of the experiments remain the same as Section 4.4.1. However, the results from Section 4.4.3 are not directly comparable with the results from this section due to the following reasons: (1) on average, the reviewers are less capable than the reviewers from Section 4.4.3 (only the tenured professors are as capable); and (2) capability score contributes differently than *stress*, *workload*, and *incentive* in computing the quality of reviews. The result is that the performance gap between the random algorithm and the rest of algorithms are significantly narrower.

Figure 4.7 shows the *score difference* graph. The greedy and round robin algorithms drop off immediately at the beginning of the graph although the greedy algorithm maintains an advantage over the round robin algorithm. The advantage of the greedy algorithm over the round robin algorithm is perhaps due to use of the two factors. Although the greedy algorithm considers both factors, the quality of reviews factor is incorrect as the `goodness` function for the greedy algorithm ignores attributes, which results in poorer performance when compared to the attributes-greedy and attributes-enhanced algorithms. The attributes-greedy and attributes-enhanced algorithms still produce assignments that

result in 100% quality reviews. At the first point of interest (around 58 submitted papers), the attributes-greedy and attributes-enhanced algorithms are no longer able to keep some reviewers from being overburdened. However, the attributes-greedy and attributes-enhanced algorithms still maintain an advantage over the greedy and round robin algorithms. At the second point of interest (around 70 submitted papers), the attributes-greedy algorithm begins to perform worse than the greedy algorithm probably because the number of assignments becomes a more important factor than the quality of reviews factor. The attributes-enhanced algorithm still maintains a small advantage over the other algorithms because it considers both factors. The round robin algorithm barely maintains an advantage over the random algorithm because it ignores the score of an agent's reviewing capability, which matters in these experiments. At the third point of interest (around 106 submitted papers), the performance of all algorithms seem to converge probably because situation is bad enough that any algorithm would perform just as well.

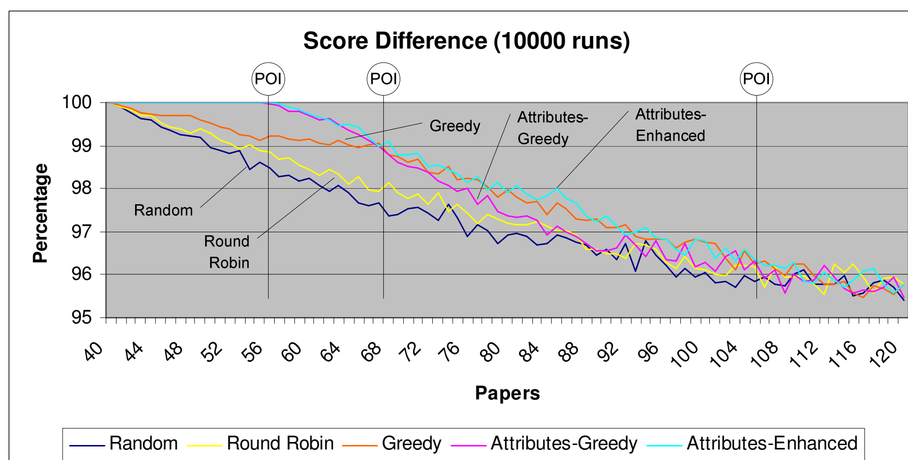


Figure 4.7: Score Difference Graph

Figure 4.8 shows the *set commonality* graph. Again, the round robin and greedy algorithms drop off immediately at the beginning of the graph but the greedy algorithm, which considers both factors, maintains an advantage over the round robin algorithm. At the first point of interest (around 56 submitted papers), the attributes-greedy and attributes-

enhanced algorithms are not able to keep some reviewers from being overburdened but they still maintain an advantage over the other algorithms. At the second point of interest (around 66 submitted papers), the greedy algorithm almost catches up to the attributes-enhanced algorithm and the attributes-greedy algorithm begins to perform worse than the greedy algorithm. This change is probably due the number of assignments factor becoming the dominant factor. At the third point of interest (around 104 submitted papers), the performance of all algorithms seem to converge probably because the situation is severe enough that any algorithm would perform just as good. Although, the attributes-enhanced algorithm seem to be slightly better the other algorithms.

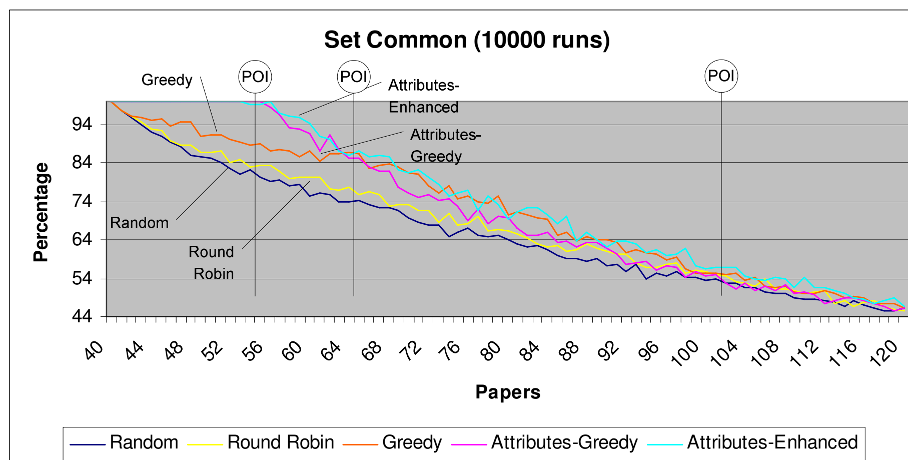


Figure 4.8: Set Commonality Graph

Figure 4.9 shows the *review quality* graph. Again, the round robin and greedy algorithms start out worse than the attributes-greedy and attributes-enhanced algorithms. However, the greedy algorithm, which considers both factors, maintains an advantage over the round robin algorithm. At the first point of interest (around 58 submitted papers), the attributes-greedy and attributes-enhanced algorithms are no longer able to keep some reviewers from being overburdened but they still maintain an advantage over the other algorithms. At the second point of interest (around 68 submitted papers), the greedy algorithm surpasses the attributes-greedy algorithm because the number of assignments factor becomes just as

important as the quality of review factor. The attributes-enhanced algorithm still maintains a slight advantage over the greedy algorithm because it considers both factors properly. At the third point of interest (around 80 submitted papers), the number of assignments factor becomes the dominant factor. This results in the attributes-greedy algorithm performing worse than the round robin algorithm. At the fourth point of interest (around 106 submitted papers), the attributes-greedy algorithm performs worse than the random algorithm. This is probably due to the overwhelming importance of the number of assignments factor over the quality of reviews factor. Also, the performance of the greedy algorithm is surpassed by the round robin algorithm probably because the greedy algorithm incorrectly considers the two factors. The attributes-enhanced algorithm maintains a slight advantage over the round robin algorithm because it considers the two factors properly.

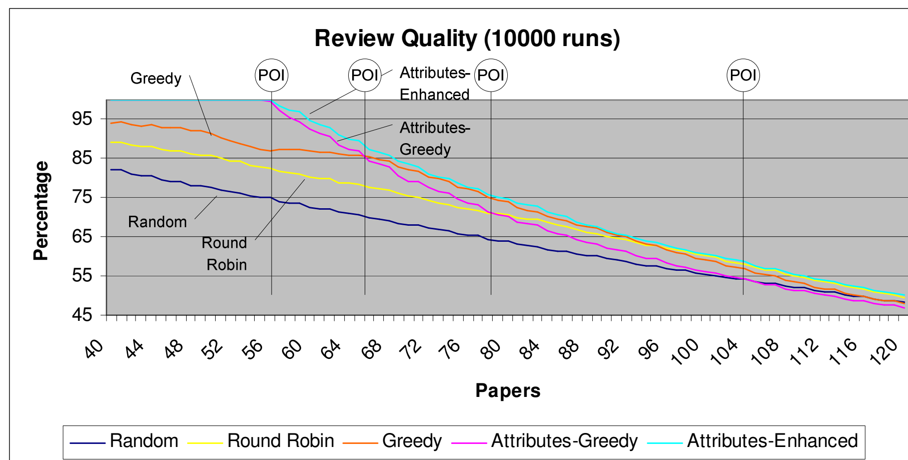


Figure 4.9: Review Quality Graph

With the introduction of attributes, algorithms that take advantage of this extra information are able to perform better. However, the caveat of this extra information is that it needs to be considered in the proper context as the attributes-greedy algorithm demonstrates through the three graphs. Also, just using the extra information but ignoring other existing information such as number of assignments can also lead to a decrease in performance as seen in the three graphs when the greedy algorithm surpasses the attributes-

greedy algorithm.

4.4.5 Time Complexity

This section discusses the time complexity of the four algorithms in Section 4.4.2. None of the algorithms reshuffles current assigned goals, the algorithms only assign goals that have not been assigned.

Let g be the number of unassigned goals, a be the number of agents in the organization, r be the number of roles in the organization, c be the number of capabilities in the organization, and n be the number of attributes in the organization. The time complexity of the random, round robin, and greedy algorithms is $O(g \times a \times r \times c)$, while the time complexity of the attributes-greedy and attributes-enhanced algorithms is $O(g \times a \times r \times (c + n))$. For a detailed proof, refer to Appendix D.1, D.2, D.3, D.4, and D.5. Introducing attributes to CzM increases the time complexity of the `goodness` function by an expected amount. Although the time complexity of the attributes-enhanced algorithm is not affected by Equation 4.20, a generalized implementation could increase the time complexity.

4.5 Multiple Humans Multiple Robots Evaluation

The scenario described in Section 4.4 has three limitations: (1) all the agents are modeled as humans, (2) the bulk of the assignments occur at the same time (i.e., when deciding the papers to be assigned to reviewers), and (3) there is only one PMF. The retrieval scenario in this section addresses the three limitations by having a mix of humans and robots, tasks that occur at different times, and multiple PMFs.

The retrieval scenario has two recurring tasks: (1) retrieve an item and (2) relay messages. The retrieve task is the primary task where performance will vary based on the agent. The relay task is a secondary task that also affects the performance of the

retrieve task. However, the performance of the relay task is constant regardless of the agent performing it. These tasks can occur at different times and in different amounts. In addition, retrieve tasks can have different minimum completion time while the completion time of relay tasks is constant. An agent can only work on one retrieve task at a time but can work on multiple relay tasks at the same time. However, human performance degrades when a human is working on too many tasks at the same time. Furthermore, as humans continue to perform their recurring tasks, their performance will degrade over time.

Figure 4.10 shows the GMoDS model that captures the retrieval scenario. There are three leaf goals, which are *retrieve*, *relay*, and *initialize*. The *retrieve* and *relay* goals represent the two recurring tasks and the *initialize* goal allows GMoDS to create new *retrieve*/*relay* goals.

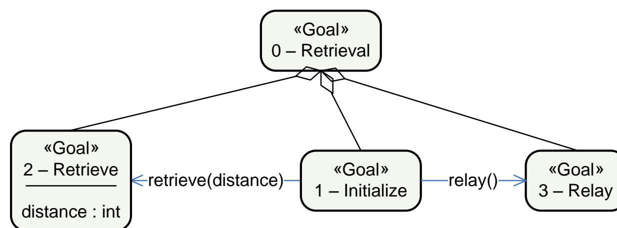


Figure 4.10: Retrieval Goal Model

Figure 4.11 shows the OMACS role model with three roles, each of which achieves a specific leaf goal. Because the CzM model also captures attributes, the *retriever* role and the *relayer* role are modified to include *fatigue* and *workload* as necessary attributes as shown in Figure 4.12. For the purposes of this experiment, all humans and robots are capable of performing the *retriever* and *relayer* roles. There is no need to have separate roles for the robots and human such as *human retriever* role and *robot retriever* role.

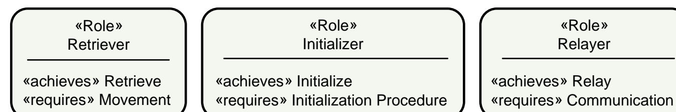


Figure 4.11: Retrieval Role Model – OMACS

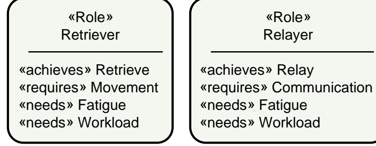


Figure 4.12: Modified Roles

The workload PMF computes the workload of a given agent, which is the sum of the workload of all the tasks that are currently assigned to the agent plus a task that may be assigned to the agent. The workload PMF is defined by two equations: Equation 4.23 defines the workload of the retrieve and relay tasks (i.e., the retrieve and relay goals respectively) and Equation 4.24 defines the workload of a given agent based on its current set of assignments plus a new task.

$$\text{workload}(g) = \begin{cases} 55\% & \text{if } g = \text{retrieve} \\ 15\% & \text{if } g = \text{relay} \end{cases} \tag{4.23}$$

$$\text{pmf}_{\text{workload}}(r, a, g, \phi) = \begin{cases} 0 & \text{if } a = \mathit{robot} \\ \left(\sum_{(a,r',g') \in \phi} \text{workload}(g')\right) + \text{workload}(g) \end{cases} \tag{4.24}$$

The fatigue PMF computes the fatigue of a given agent, which is the sum of the fatigue at the completion of all currently assigned tasks plus a task that may be assigned to the agent. The fatigue PMF is defined by two equations: Equation 4.25 defines the fatigue at the completion of the retrieve and relay tasks and Equation 4.26 defines the fatigue of a given agent based on its current set of assignments plus a new task.

$$\text{fatigue}(g) = \begin{cases} 3\% \times g.\text{distance} & \text{if } g = \text{retrieve} \\ 2\% & \text{if } g = \text{relay} \end{cases} \tag{4.25}$$

$$\text{pmf}_{\text{fatigue}}(r, a, g, \phi) = \begin{cases} 0 & \text{if } a = \text{robot} \\ a.\text{fatigue} + \left(\sum_{(a,r',g') \in \phi} \text{fatigue}(g')\right) + \text{fatigue}(g) & \end{cases} \tag{4.26}$$

4.5.1 Experimental Setup

The experiment was designed to evaluate the performance of the CzM model versus the OMACS model using two brute force task allocation algorithms on the retrieval scenario. There are several parameters for the retrieval scenario: the number of agents, the number of tasks, a given time range in which the recurring tasks can appear, and a given range for the distance parameter of the retrieval goal. The retrieval experiments were set up such that the numbers of agents was fixed at 6, the time range at which tasks can appear was $[1, 15]$, and the range for the distance parameter was $[1, 10]$. There were a total of 10 experiments. The first experiment started at 5 retrieval tasks, the second at 10 retrieval tasks, the third at 15 retrieval tasks, and so forth, up to the 10th experiment with 50 retrieval tasks. The purpose of the experiment was to evaluate how well the two task allocation algorithm assigned the recurring tasks so that the time taken to complete all tasks is as short as possible.

Unlike the algorithms from Section 4.4.2, which are greedy in nature, the two algorithms used for the retrieval scenario were brute force algorithms that go through all combinations of assignments to find the best one. Because of the brute force nature of the two algorithms, their time complexity is exponential (further discussion on the time complexity is in Section 4.5.4).

There were three types of agents: capable humans, average humans, and robots. These types capture the differences in ability when performing the retrieval task; capable humans were the best, followed by average humans, and finally the robots. However, the performance

of human agents were affected by fatigue and workload, whereas the performance of the robots were consistent. As fatigue increased, human agents took longer to complete retrieval tasks. Similarly, as workload increased (beyond a threshold), human agents took longer to complete retrieval tasks. All three types of agents were equally capable of performing the relay tasks and the time taken to complete relay tasks was constant. Table 4.3 shows the starting values of the two attributes for the three types of agents and the two capabilities required by the two recurring tasks.

	Fatigue	Workload	Retrieval Ability	Relay Ability
Capable Humans	0%	0%	1.0 (100%)	1.0 (100%)
Average Humans	0%	0%	0.75 (75%)	1.0 (100%)
Robots	0%	0%	0.5 (50%)	1.0 (100%)

Table 4.3: Attribute and Capability Values of Agent Types

Performance for the retrieval task was based on fatigue, workload, ability, and the distance parameter of the retrieval task. As a baseline, a perfect agent (1.0 ability, 0% fatigue and workload, and is unaffected by fatigue and workload) would complete a retrieval task in d time units, where $d = g.\text{distance}$. In general, Equation 4.27 defines the progress that an agent made in one time unit when performing the retrieval task.

$$\begin{aligned}
&d\Delta = \text{distance} * (1 - \text{ability}) \\
&f\Delta = \text{distance} * \text{fatigue} \\
&w\Delta = \text{distance} * \max(\text{workload} - 1, 0) \\
&\text{estimated completion time} = \text{distance} + d\Delta + f\Delta + w\Delta \\
&\text{progress} = \frac{1}{\text{estimated completion time}}
\end{aligned} \tag{4.27}$$

For example, if a capable human was given a retrieval task with 2 distance. Then the progress made in the first time unit was $\frac{1}{2}$ because fatigue is 0 and workload was less than 1. However, in the next time unit, the fatigue of that agent increased by 3% (Equation 4.25). So the progress made for that task in the second time unit was $\frac{1}{2.06}$ and the fatigue of the

agent increased by $\frac{1}{2.06} \times 2$ distance $\times$ 3% fatigue $\approx$ 2.91%. But the total progress of the task was only $\approx$ 98.5% complete, so the agent had to take a third time unit to complete the task, at which point the fatigue of the agent increased by $\approx$ 0.09% to a final value of 6% fatigue.

There were six agents for every experiment: two capable humans, two average humans, and two robots. The reason the number of agents was small is because of the highly exponential time complexity of the brute force algorithms. It was not possible to obtain data in a reasonable amount of time if the number of agent was larger than 10.

Due to the randomness of the experiments (the time in which the two recurring tasks can appear and the distance for the retrieval tasks) and the exponential time complexity, each experiment was only executed 1000 times to normalize the data.

4.5.2 Algorithms

The two brute force algorithms are essentially the same, the only differences were in computing the score for an assignment and the overall score. With a given list of unassigned goals, the two brute force algorithms computed a mapping for each unassigned goal. The mapping is from an unassigned goal to a list of agents capable of achieving that goal. An example is shown in Table 4.4 with four goals (g_1, g_2, g_3, and g_4) and four agents (a_1, a_2, a_3, and a_4). For instance, g_3 can be assigned to either a_3 or a_4.

g_1	g_2	g_3	g_4
a_1	a_2	a_3	a_4
a_2	a_3	a_4	
a_3	a_4		
a_4			

Table 4.4: Example Table

Once the table is computed, the two brute force algorithms go through every permutation to create an assignment set. Using the example from Table 4.4, there are 24

combinations of an assignment set. For example, a combination of an assignment set is $\{\langle g_1, a_1\rangle, \langle g_2, a_2\rangle, \langle g_3, a_3\rangle, \langle g_4, a_4\rangle\}$. Another example is $\{\langle g_1, a_1\rangle, \langle g_2, a_2\rangle, \langle g_3, a_4\rangle, \langle g_4, a_4\rangle\}$. The two algorithms compute the overall score for each assignment set and pick the assignment set with the highest score as the best one.

In the OMACS version of the algorithm, the computation of the score for an assignment is defined by Equation 4.28. The computation of the overall score for a set of assignments is defined by Equation 4.29, where assigned(Φ) is the set of agents that are currently assigned, total(a, Φ) is the number of assignments for agent a, score(a) is the retrieval task score for agent a, relays(a, Φ) is the number of relay tasks assigned to agent a, and relays(Φ) is the number of all relay tasks.

$$\text{score} = \sqrt[|\{c|(r,c)\in\text{requires}\}|]{\prod_{c\in\{c|(r,c)\in\text{requires}\}} \text{possesses}(a, c)} \tag{4.28}$$

$$\sum_{a\in\text{assigned}(\Phi)} \left(\frac{1}{\text{total}(a, \Phi)}\right) \text{score}(a) - \text{relays}(a, \Phi) \left(\frac{\text{relays}(a, \Phi)}{\text{relays}(\Phi)}\right) \left(\frac{\text{relays}(a, \Phi)}{\text{total}(a, \Phi)}\right) \tag{4.29}$$

In the CzM version of the algorithm, the computation of the score for an assignment depends on the task type. If the task is the retrieval task then the score computation is defined by Equation 4.30. If the task is the relay task then the score computation is defined by Equation 4.31. The computation of the overall score for a set of assignments is defined by Equation 4.32, where score(a, r, g) is the score for the assignment.

$$
\begin{aligned}
&d\Delta = \text{distance} * (1 - \text{ability}) \\
&f\Delta = \text{distance} * \text{pmf}_{\text{fatigue}}(r, a, g, \phi) \\
&w\Delta = \text{distance} * \max(\text{pmf}_{\text{workload}}(r, a, g, \phi) - 1, 0) \\
&\text{estimated completion time} = \text{distance} + d\Delta + f\Delta + w\Delta \\
&\text{score} = \frac{\text{distance}}{\text{estimated completion time}} + \text{distance}
\end{aligned}
\tag{4.30}
$$

$$
\begin{aligned}
f\Delta &= \frac{1}{\text{pmf}_{\text{fatigue}}(r, a, g, \phi) + 1} \\
w\Delta &= \min(\frac{1}{\text{pmf}_{\text{workload}}(r, a, g, \phi)}, 1) \\
\text{score} &= \frac{f\Delta + w\Delta}{2}
\end{aligned}
\tag{4.31}
$$

$$
\text{overall score} = \sum_{(a,r,g) \in \Phi} \text{score}(a, r, g) \tag{4.32}
$$

4.5.3 Results

Two types of data were collected in the experiments: *cases* and *completion time*. *Cases* measured the number of runs in which the CzM algorithm performed worse than, equal to, or better than the OMACS algorithm. *Completion time* measured the average time in which a run took to complete all tasks.

Figure 4.13 shows the results of the two algorithms as measured by *cases*. When the number of retrieval tasks is low, there was a small number of cases where the OMACS algorithm performed better, a significant number of cases where the two algorithms had the same performance, and a small number of cases where the CzM algorithm performed better. However as the number of retrieval tasks increases, the trend changes. When the number of retrieval tasks is over 20, in virtually every case, the CzM algorithm performed better than the OMACS algorithm.

There are two graphs for the *completion time* data. The first figure, Figure 4.14, shows

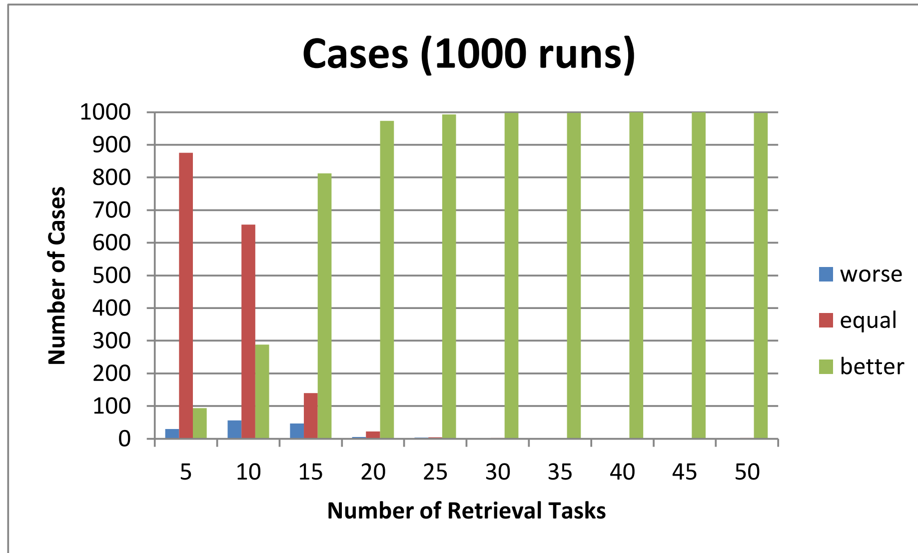


Figure 4.13: Cases Graph

the results of the two algorithms as measured by average *completion time*. In addition to the results from the two algorithms, a third line is added to the graph. The third line is the completion time using only six perfect agents; perfect agents are unaffected by fatigue and workload and have the best ability when performing retrieval tasks. The reason for the third line is to provide an approximation of the lower bound for the completion time. Ideally, the third line should come from an optimal algorithm. However, due to the exponential time complexity, it would take too long to obtain the results. For example, on a small case (where there are 3 agents, 12 relay goals, and 2 retrieval goals), there are approximately about 4 million (3^{12+2}) paths to explore per run. A path takes about 10 seconds[5] to explore, so exploring all 4 million paths would take about 1.2 years. However, the perfect agents line is a loose approximation because agents never tire or drop in their performance whereas the performance of human agents continue to deteriorate the more tasks they perform, more so towards the higher end (50 retrieval tasks). Even an optimal algorithm (with normal agents) cannot not perform better than the perfect agents line. Furthermore, the difference

[5]10 seconds is an estimate for an 8-core Intel® Xeon® E5462 @ 2.80GHz with 12GB RAM.

between the optimal line and the perfect line should be increasing as the number of retrieval tasks increases.

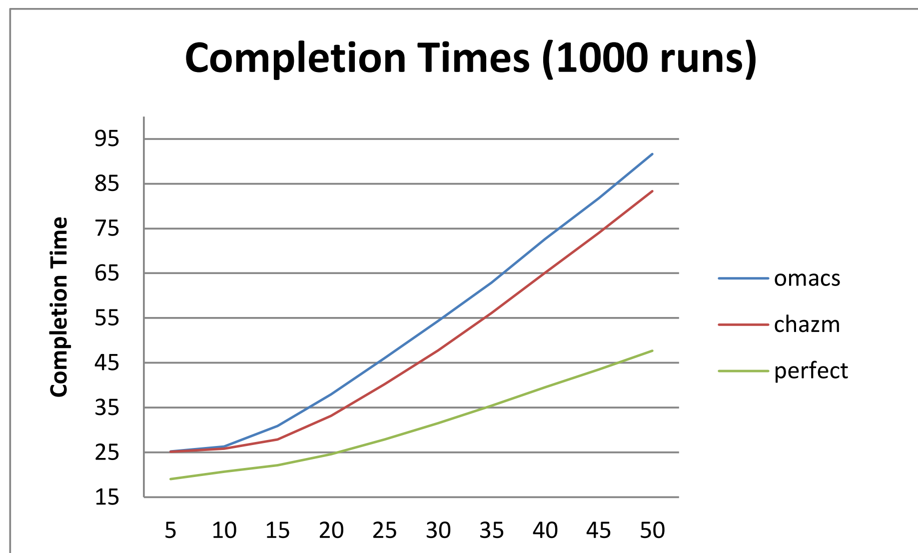


Figure 4.14: Completion Time Graph

The second graph, Figure 4.15, shows the average *completion time* along with one standard deviation from the average for the OMACS and CzM models. There is an overlap between the two models, which occurs between the averages of the two models. Even though there is an overlap, the conclusions drawn from Figure 4.13 are still valid. The overlap occurs because of the random distances for the retrieval tasks and the times at which tasks appear. For example, using 5 retrieval tasks, there can be a case where all 5 retrieval tasks have the same starting time with all of them having a distance of 1. In another case, the 5 retrieval tasks can have the same starting times as the previous case but a distance of 10. This wide range of possible values creates a large variation of possible completion times. However, as shown in Figure 4.13, in the same case, the CzM model is usually better (after 15 retrieval tasks) or the same (before 15 retrieval tasks) as the OMACS model.

When the number of retrieval tasks is low, there is almost no difference between the two algorithms. However, as the number of retrieval tasks increases, the difference between the

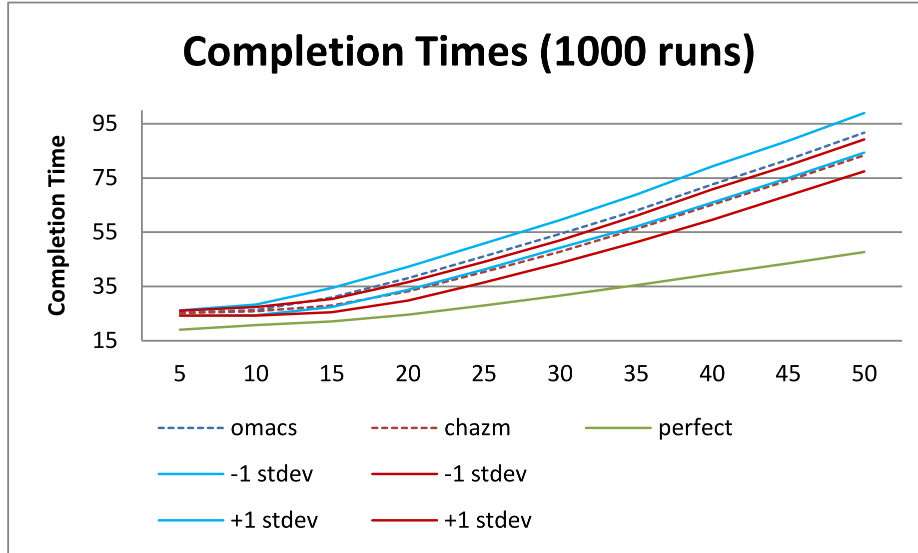


Figure 4.15: Completion Time Graph (±1 Standard Deviation)

two algorithms becomes noticeable. The CzM algorithm maintains a noticeable difference ($\approx$ 10% difference in terms of completion time) over the OMACS algorithm.

The results of this evaluation show that attributes and PMFs in the CzM model can allow continuous task allocation algorithms to perform better when a mix of humans and robots are involved. The OMACS algorithm is already very good (within $\approx$ 80% of the perfect line at the early part of the results) and the CzM algorithm (an improvement of $\approx$ 10% over the OMACS algorithm) is also better in virtually every case when there are over 20 retrieval tasks.

4.5.4 Time Complexity

This section discusses the time complexity of the two brute force algorithms in Section 4.5.2. The two algorithms only assign goals that have not been assigned.

Let g be the number of unassigned goals, a be the number of agents in the organization, c be the number of capabilities in the organization, and n be the number of attributes in the organization. The time complexity of both brute force algorithms is $O(a^g \times (g \times (c + n)))$,

where $n = 0$ for the OMACS version. For a detailed proof, refer to Appendix D.6. The use of attributes increases the time complexity by an expected amount.

4.6 Summary

This chapter presents the CzM model and demonstrates that task allocation algorithms can benefit by including human performance factors, which are captured as attributes. The results from the first experiment (Section 4.4) show that in a system consisting of all humans, the use of attributes in bulk task allocation algorithms can produce better results. The second experiment (Section 4.5) shows that in a system consisting of a mix of humans and robots, the use of attributes in incremental task allocation algorithms can produce better results. The next chapter (Chapter 5) describes another aspect of human integration: *organization control*.

Chapter 5

Organization Control

This chapter introduces an alternative interaction scheme to the playbook style [44, 63] of supervisory control [16]. In Chapter 1, it was mentioned that traditional supervisory control is not highly scalable because according to the evaluation system proposed by Crandall *et al.* [18], there is a low upper bound to the number of robots a human supervisor can handle [17]. Figure 1.2 illustrates the general approach to addressing this scalability limitation. One particularly popular approach is the playbook style. However, in playbook style approaches, the human selects a group of robots and the appropriate play for that group. Unfortunately, playbook style is not appropriate because a play is typically predefined, which includes a fixed number of players.

Using the example from Chapter 2, suppose that the commander wants John and Surveyor 1, who are currently surveying $A1$ as a team, to also survey $A2$. It would be awkward if the commander has to issue the order to each member of the team (i.e., issuing the order to John to survey $A2$ and issuing the same order to Surveyor 1 to survey $A2$). In fact, it is intuitive for the commander to issue the order to the team once instead of individually to each member. From the above example, the commander issues a high-level goal to the team and it is up to the team to adjust appropriately to the goal. The intent is

to regard groups as part of an organizational hierarchy and through the organization allow a human supervisor to exercise supervisory control (*organization control* is supervisory control over an organization).

First, a broad definition of an organization. An *organization* is a group that is working towards a common goal [8]. An *organization* is composed of three basic entities: *goals*, *roles*, and *agents*. *Goals* define the purpose and intent of the organization. *Roles* form the behavioral how-to to achieving those goals. And *agents* are the entities that carry out their respective roles to achieve those goals. In any organization, a *task* is defined as the pairing of a goal with a role and the pairing of an agent with a task is defined as an *assignment*. Next, is the definition of organization control.

Definition 5.1. Organization control ...

1. exists as a single mechanism that a human supervisor interacts with to exercise supervisory control over an organization.

2. defines a consistent set of interactions.

3. contains a mechanism that makes decisions autonomously but still allows a human supervisor to modify those decisions.

To achieve organization control, the human supervisor should not need to interact with the agents individually. In fact, most (if not all) of the interactions with individual agents should be relegated to an autonomous mechanism. This mechanism should be an abstraction of all the agents in the system. By eliminating the need for a human supervisor to interact with agents individually, the number of agents should no longer be a limitation. However, just because the human supervisor no longer needs to interact with the agents individually does not mean that the direct interactions with agents are no longer necessary. These direct interactions are still required but instead of the human

supervisor interacting with the agents, these direct interactions are handled by a layer that sits between the human supervisor and the agents. Furthermore, to facilitate control over an arbitrary number of agents, this layer must also be able to handle more complex functions of autonomy to alleviate the human supervisor from dealing with the agents individually. One such autonomous function is autonomous task allocation; another autonomous function is autonomous tracking of the current set of tasks and determining the next set of tasks. Figure 5.1 illustrates the concept of organization control through this layer, which is named the Intelligent Autonomous Layer (IAL).

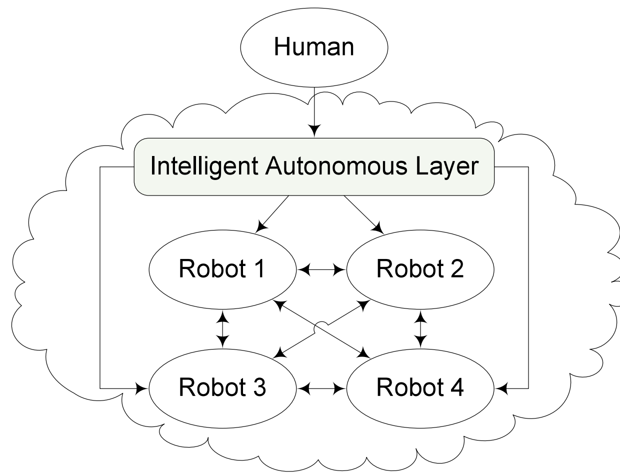


Figure 5.1: Organization Control

The human supervisor interacts with the IAL while the IAL handles the individual interactions with agents such as informing them of their assignments. Furthermore, the IAL incorporates some decision-making processes. One such process is task allocation; the IAL autonomously decides "who is working on what". First, the set of interactions that can occur between a human supervisor and the IAL for organization control are defined in Section 5.1. The following section (Section 5.2) describes an agent architecture that facilitates implementation of the IAL. The next two sections describe two of the interactions for organization control in detail. Section 5.3.1 describes how assignment set manipulation works and Section 5.3.2 describes an extension to GMoDS [66] to allow goal modification.

And Section 5.4 summarizes this chapter.

5.1 Interactions of Organization Control

This section explores a set of interactions that can occur between a human supervisor and the IAL for organization control. Based on the broad definition of an organization (at the beginning of this chapter), the following is a list of descriptions of the interactions that a human supervisor should be able to perform on an organization. Since organization control is mostly about control over an organization, the focus is on interactions with the organization rather than individual members. However, there are some individual interactions that still persist at the organization-level because of assignments (i.e., "who is working on what"). Even though assignment is an organizational concept, manipulation of assignments can still be considered an individual interaction because the human supervisor is essentially making the decision on "who is working on what"; thus, manipulating assignments is no different from traditional supervisory control. Furthermore, the set of interactions should be available through a single mechanism such as the IAL. The following list is the set of interactions for organization control.

Create Goals. When an organization interacts with its environment and pursues its goals, that organization is likely to evolve over time. Thus, a human supervisor should be able to add goals to the organization. These goals can be more of the same types of goals or new goal types that are not part of the initial design. Using the example from Chapter 2, suppose that a team has already completed surveying $A1$ but the commander wants that area to be surveyed again by another team. So, the commander creates another *survey A1* goal. In another situation, suppose that the team surveying $A2$ gets trapped by crumbling debris. So, the commander creates a new goal type, which is to clear the debris to free trapped team.

Remove Goals. Likewise, as organizations evolve, there are going to be goals that are no longer necessary. Thus, a human supervisor should be able to remove goals from the organization. Using the example from Chapter 2, suppose that before area $A4$ can be surveyed, that area is completely closed off due to further debris collapse. Since the area $A4$ is no longer accessible, the goal *survey A4* is no longer necessary and can be removed by the commander.

Modify Goals. Similarly, as organizations continue to evolve, the objectives of an organization could evolve and so there are going to be goals that need to change over time to reflect the changes in an organization. Thus, a human supervisor should be able to modify existing goals. Using the example from Chapter 2, suppose that a search area is defined as regions that should be explored and surveyed. A sudden collapse of a ceiling could close off a portion of an existing region from being explored and surveyed. For these type of changes, only the boundaries that define the affected region need to be changed and only the affected goal needs to change. Rather than removing the affected goal and creating a new one, it is more effective to just change the affected goal. In this case, it would be easier for the commander to modify the affected goal instead of removing it and creating a new one.

Achieve Goals. As organizations continue to pursue their goals, these goals may eventually be achieved. Thus, a human supervisor should be able to mark goals as achieved. Using the example from Chapter 2, once the team assigned to the goal *survey A1* completes it, that goal is now achieved. However, the commander can also manually achieve the *survey A1* goal if the commander decides that enough effort has been expended on the goal.

Fail Goals. Likewise, an organization could fail to achieve some of their goals. Thus, a human supervisor should be able to mark goals as failed. Using the example from

Chapter 2, suppose that the commander imposes a time limit on the *survey A1* goal and that the team assigned to that goal has already exceeded the allotted time; that means that the goal has failed and the commander should mark that goal as failed.

Create Roles. When new goal types are created that are not part of the original design, these new goals may require new roles to capture the behavioral requirements to achieve the new goals. Thus, a human supervisor should be able to create new roles. Using the example from Chapter 2, when the commander creates a new goal type to clear debris to free a trapped team, a new role needs to be created to capture how that new goal can be achieved because no existing role can achieve the new goal.

Remove Roles. Likewise, when goal types are removed, the accompanying roles may no longer be necessary. Instead of leaving them orphaned in the organizational structure, they should be removed. Thus, a human supervisor should be able to remove existing roles. Using the example from Chapter 2, suppose that there is no need to identify hazards because prior to the earthquake, all hazardous materials were removed from the building. In this case, the commander may remove the role for identifying hazards from the organizational structure because there is not going to be any hazards that need identification.

Modify Roles. When organizations evolve, the behavior associated with roles may change. This change can be a result of external influences such as changes to existing laws that affect how an organization should act. Sometimes, such changes require modification to the behavior of the roles. Thus, a human supervisor should be able to modify existing roles. Using the example from Chapter 2, suppose that there is a change to how Cardiopulmonary Resuscitation (CPR) is performed. If mouth-to-mouth is no longer part of the process, the role for rescuing victims would require an update to the CPR behavior.

Add Agents. Even if the goals and roles of an organization do not change, the set of agents may not remain the same throughout the life of the organization. Thus, the human supervisor should be able to add agents to the organization. Using the example from Chapter 2, suppose that the commander requests assistance from other emergency response teams. When help arrives, the commander will need to add these new members (agents) to the organization.

Remove Agents. Similarly, agents also leave organization for various reasons. Thus, a human supervisor should be able to remove agents from the organization. Using the example from Chapter 2, suppose that John is injured and is sent to a hospital. Since he can no longer continue to function as part of the organization, the commander should remove John from the organization.

Create Assignments. Assignments are an important part of the organizational structure as they indicate "who is working on what". Thus, a human supervisor should be able to create new assignments when necessary. Using the example from Chapter 2, suppose that the commander decides to add an extra person to the team currently surveying $A1$, the commander must also create a new assignment for that person.

Remove Assignments. Similarly, throughout the life of an organization, the "who is working on what" changes over time. Thus, a human supervisor should be able to remove assignments when necessary. Using the example from Chapter 2, suppose that the commander notices that John of the team currently surveying $A1$ is extremely fatigued and also injured. But there is a new fresh member, Alison, who is currently doing nothing. The commander decides to replace John with Alison. Thus, the commander must remove the assignment for John.

The set of interactions defined above are the interactions for organization control that deals with the goals, roles, agents, and assignments. When dealing with agents, a human

supervisor can "add" (not "create" as the goals, roles, or assignments) because agents can be either humans, robots, or software agents. Only in the case of software agents can a human supervisor spawn a process for new agents and even then, the human supervisor may not have the authority to spawn new processes. More so for human/robot agents, a human supervisor does not have the power to magically "create" new humans or robots. Thus, a human supervisor can only add agents. Furthermore, a human supervisor cannot "modify" agents as the human supervisor typically does not have total control over agents.

Also, the "modify" assignment interaction is missing because there is no semantic difference between a "modify" operation and using a "remove" assignment interaction followed by a "create" assignment interaction. An assignment modification would mean a change either the agent, role, or goal of the assignment such as changing the role to a different one. Any of the three changes would require the agent to be informed to stop working on the current assignment and the new assignment. Removing the assignment would perform the notification, and creating the new assignment would perform the notification.

The next section (Section 5.2) describes a approach for including organization control in multiagent systems. This approach creates the IAL, which is the mechanism with which a human supervisor interacts to achieve organization control.

5.2 Intelligent Autonomous Layer (IAL)

The IAL (as shown in Figure 5.1) is the mechanism that sits between a human supervisor and a group of agents. The human supervisor interacts with the IAL while the IAL interacts with all the agents. Furthermore, it is through the IAL that a human supervisor can exercise organization control. The IAL can be formed through the Organization-Based Agent Architecture (OBAA) [23]. So, the first step is to explain the OBAA architecture. The OBAA is an architecture for implementing agents. The OBAA provides a separation

between the application specific and non-application specific implementations. Figure 5.2 illustrates the architecture, where an OBAA agent is comprised of two major components: the Execution Component (EC) and the Control Component (CC). Generally, the EC contains the application specific implementation of an agent such as role behaviors and capabilities. On the other hand, the CC contains the non-domain specific and non-application specific implementation (i.e., the CC contains general algorithms that use application specific information).

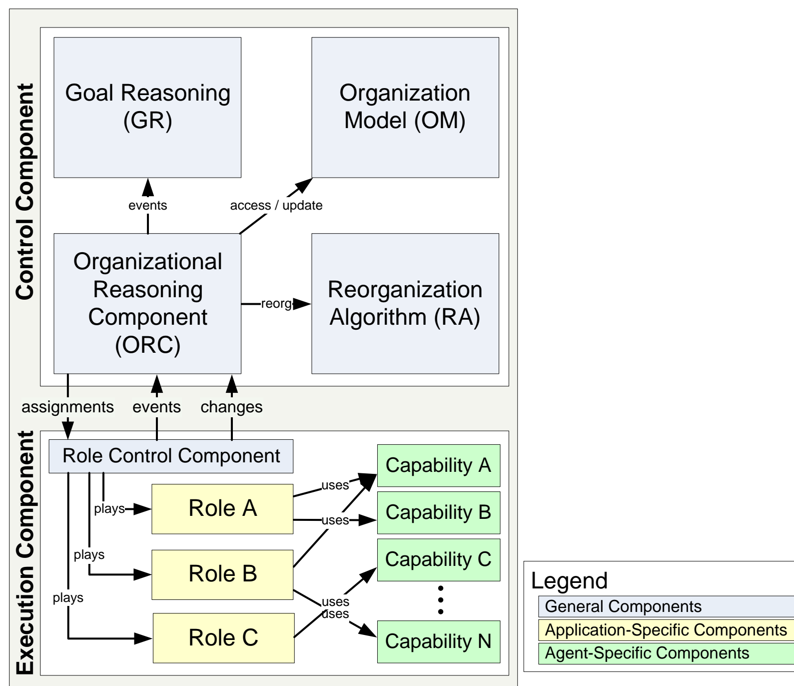


Figure 5.2: Agent Architecture

The EC consists of three components: the Role Control Component (RCC), a set of roles, a set of capabilities, and a set of attributes. The RCC functions as the interface between the EC and the CC and as a scheduler for the assignments that have been given to the agent. The RCC handles the assignments that are passed from the CC, which determines which roles to use to complete the assignments. Furthermore, the RCC informs the CC of events that occur while performing roles and changes to the agent's capabilities and attributes.

The CC consists of four components: the Goal Reasoning (GR), the Organization Model (OM), the Reorganization Algorithm (RA), and the Organizational Reasoning Component (ORC). The GR performs goal-based reasoning such as deciding the sequence of goals to achieve and modifying the set of goals based on events that occur during the execution of the system. The OM is the knowledge repository that contains information about the current structure of the organization such as the agents that are present in the organization, the goals that the system is pursuing, and the set of assignments. The RA is the assignment algorithm that computes the initial set of assignments as well as subsequent assignments due to failures and changes to agents' capabilities and/or attributes. It is possible to include application-specific assignment policies into the RA such as "an agent can be assigned at most one task". The ORC is the interaction point between the EC and CC where assignments are handed out to the EC and events received from the EC are processed. The ORC performs three important functions. First, the ORC decides when and how events are processed. Second, the ORC decides when and how to reorganize. Reorganization means changing the set of assignments. For example, when an agent's capabilities degrades to a point where it is no longer capable of carrying out its current assignment, the ORC uses the RA to determine a new assignment. Third, the ORC is responsible for maintaining accurate information about state the agents and the overall progress of the system. The IAL is composed of the CC across all agents working together as illustrated in Figure 5.3. To human supervisors, the objective is to make it appear as if there is only one CC in the system.

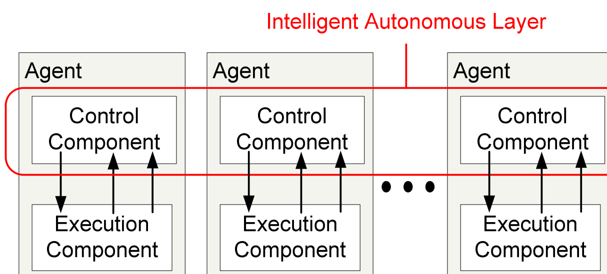


Figure 5.3: Intelligent Autonomous Layer (IAL)

It is when the human supervisor is interacting with the IAL that the human supervisor can exercise organization control. By exercising organization control, the burden of dealing with robots individually now rests with the IAL instead of the human supervisor. Due to the way the CC is structured, the set of interactions that is available for use by a human supervisor depends on whether the GR and the OM support the interactions.

First, there is a need to evaluate the useful of the IAL before proceeding further with organization control. The next section (Section 5.2.1) looks at how the IAL can perform its autonomous function of task allocation. Autonomous task allocation frees up the human supervisor from micromanaging the "who is working on what". Furthermore, the IAL also inherits from GMoDS the ability to autonomously tracks the current set of goals and determines the next set of goals based on events that occur in the environment.

5.2.1 Scenario

To evaluate the IAL, a military-based scenario is devised. In military situations, routes used for convoys must be constantly monitored for safety, including detection of Improvised Explosive Devices (IEDs) [56], which are easily disguised and hard to spot as shown in Figure 5.4. Furthermore, as IEDs can be remotely detonated, monitoring an area for safety is a high risk situation when humans are involved.

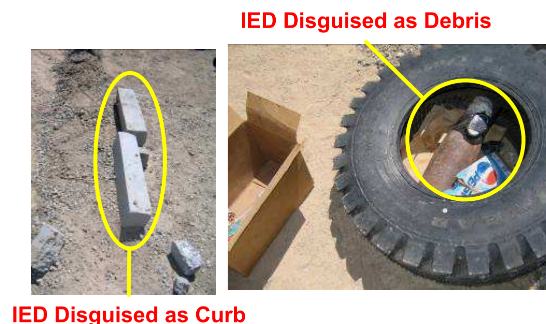
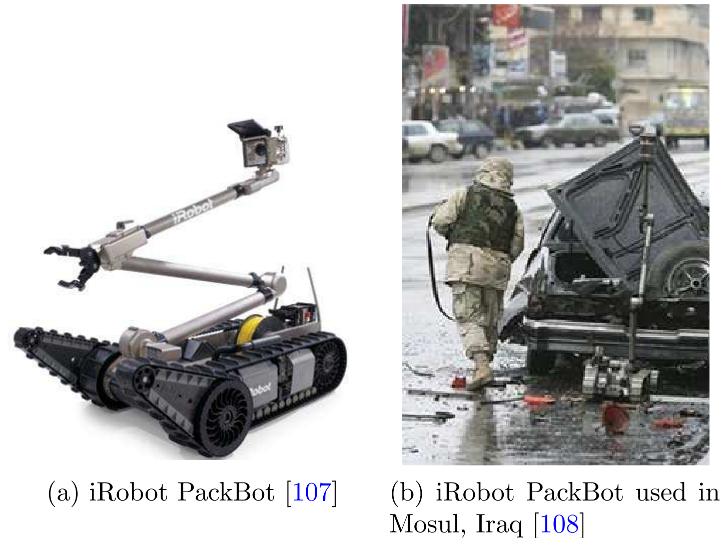


Figure 5.4: Improvised Explosive Devices

Currently, the United States Marine Corp (USMC) deploys teleoperated iRobot PackBot

[107] to safely identify and disarm IEDs as shown in Figure 5.5. Scenarios like this are well-suited for testing an autonomous system's ability to adapt to failures.



(a) iRobot PackBot [107] (b) iRobot PackBot used in Mosul, Iraq [108]

Figure 5.5: Using iRobot PackBot

A scenario is defined where two routes intersect and are to be monitored for IEDs. A human operator provides the initial input of areas to monitor and provides expertise on IED identification. The area includes the two intersecting routes and the area surrounding those routes. The IED detection system utilizes a team of heterogeneous robots that will proceed to their given areas to search for IEDs. Once the area of to monitor is given, it is partitioned into smaller areas, which are assigned to robots that are capable of detecting suspicious objects. These robots continuously monitor their given area(s) for suspicious objects. When suspicious objects are detected, they are flagged for identification and robots that are capable of identifying IEDs (which may be the same robot that detected them) are assigned to identify them. In cases when identification is not possible by the robots, a human is asked for help with the identification process. When IEDs are identified, robots that are capable of defusing or disposing of IEDs are assigned to deal with the IEDs.

Following the O-MaSE design process produced four models: a goal model, a role

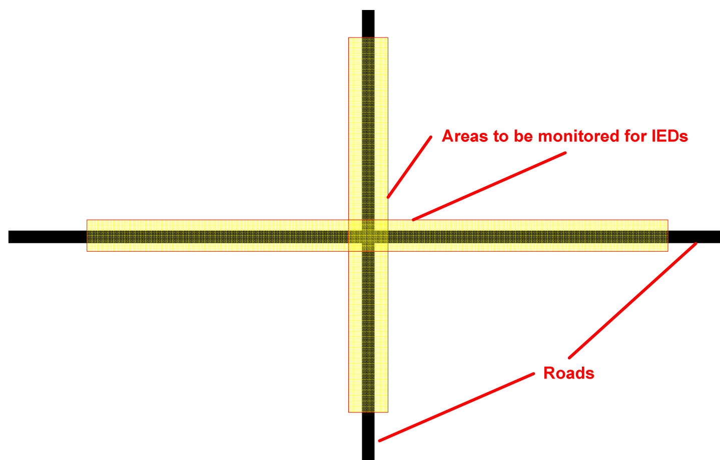


Figure 5.6: Scenario

model, a capability model, and an agent model. The goal model represents the high-level requirements of the IED scenario as well as their decomposition into the goals that are necessary to achieve them. The role model specifies the necessary behavior and capabilities required to achieve the leaf goals from the goal model. The capability model specifies the necessary functionality for every capability in the role model. The agent model defines several types of agents based on the capabilities that they possess, which are defined in the capability model.

5.2.1.1 Goal Model

Figure 5.7 shows the GMoDS goal model of the IED detection system. The top-level goal is *monitor IEDs*, which has four subgoals: *interact with user*, *monitor area*, *identify object*, and *defuse IED*. At initialization, the subgoal *interact with user* is the only goal that exists; the rest of the subgoals must be triggered by events. The *interact with user* is automatically assigned to the appropriate agent to pursue.

The *monitor area* goal is triggered by the *monitor* event while the agent is pursuing the *interact with user* goal, which also triggers the *divide area* goal as well. A *monitor* event occurs when the human operator specifies an area to monitor for IEDs. Once the

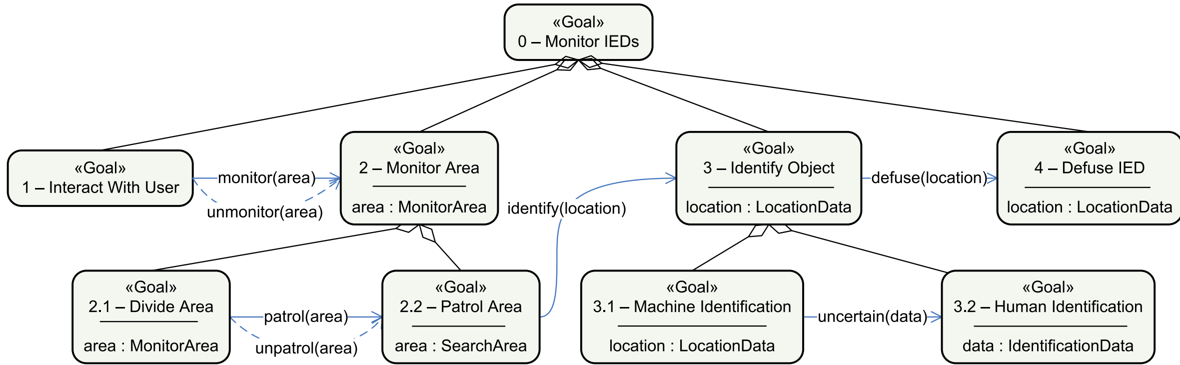


Figure 5.7: IED Goal Model

divide area goal is triggered, it is automatically assigned to the appropriate agent. The agent pursing the *divide area* goal raises the *patrol* event whenever a new area needs to be patrolled. The *patrol* event causes the *patrol area* goal to be triggered, which is automatically assigned to the appropriate agent. When the agent pursuing the *patrol area* goal detects a suspicious object, the *identify* event is raised, which causes the *identify object* goal to be triggered as well as the *machine identification* goal. The *machine identification* goal is then automatically assigned to an appropriate agent.

When the agent pursuing the *machine identification* goal is unable to successfully identify the suspicious object, it raises the *uncertain* event. The *uncertain* event causes the *human identification* goal to be triggered. The *human identification* goal is then assigned to a human expert to aid in the identification process. The *defuse* event can be raised by either the human expert or the agent pursuing the *machine identification* goal when the suspicious object is identified as an IED. The *defuse* event causes the *defuse IED* goal to be triggered which is then automatically assigned to the appropriate agent.

As shown in Figure 5.7, the goal model supports two of the interactions defined for *organization control* to a limited degree: (1) creation of the *monitor area* goal, and (2) deletion of the *monitor area* goal. This is achieved by the *interact with user* goal having a positive trigger and a negative trigger to the *monitor area* goal. However, a human

operator cannot create/delete *patrol area* goals, *identify object* goals, and *defuse IED* goals. One way to enable it is to create positive and negative triggers from *interact with user* goal to the other goals. This approach would create three new positive trigger and three new negative trigger. Furthermore, this approach would require explicit inclusion in the designs of GMoDS models to support goal creation and deletion. Instead, GMoDS can be extended to properly support some of the interactions defined for *organization control*. Section 5.3.2 describes an extension to GMoDS to support goal modification.

5.2.1.2 Role Model

Figure 5.8 shows the roles that are defined for the IED detection system, the leaf goals that the roles achieve, the capabilities that are required by the roles, and the protocol that the roles use for communication. Six roles are defined to achieve exactly one of the six leaf goals from Figure 5.7. One protocol is defined; the *inform* protocol is a simple protocol that notifies the *user interaction* role on what information to display to the human operator.

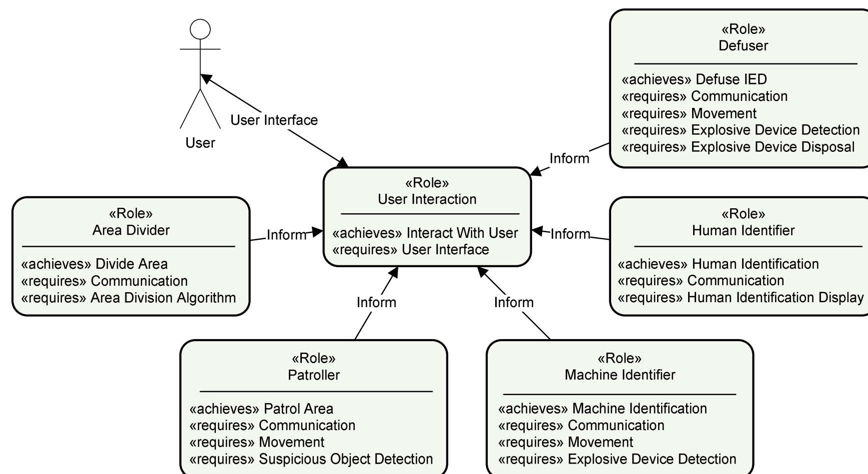


Figure 5.8: IED Role Model

The *user interaction* role specifies the functionality for a human supervisor to input the areas to be monitored for IEDs, provide the human supervisor with information about

the state of the system such as displaying search area(s), locations of agents, and current assignments. Domain-specific information comes from the *inform* protocol.

The *area divider* role handles the task of partitioning larger areas into smaller more manageable search areas. When a search area is determined, the *patrol* event occurs, which also triggers the *patrol area* goal. In addition, the *user interaction* role is informed of new search areas through the *inform* protocol.

The *patroller* role patrols a given search area for suspicious objects. Information about the search area is obtained from the *patrol area* goal when it is assigned to an agent. When a suspicious object is detected, the *identify* event occurs, which triggers the *identify object* goal. Furthermore, every time the *identify* event occurs, information about the suspicious object is passed to the *user interaction* role through the *inform* protocol.

The objective of the *machine identifier* role is to perform a more accurate analysis of a suspicious object. The location of the suspicious object is obtained from the *machine identification* goal when it is assigned to an agent. When classification of a suspicious object is not possible due to failure in meeting the accuracy requirement, the *uncertain* event occurs, which triggers the *human identification* goal. On the other hand, a suspicious object can be classified as either an IED or inert. When a suspicious object is classified as an IED, the *defuse* event occurs, which triggers the *defuse IED* goal. Otherwise, that suspicious object is classified as inert. In any case, the *inform* protocol is used to inform the *user interaction* role about the suspicious object.

The purpose of the *human identifier* role is to present sufficient information to the human peer so that the human peer can determine if a suspicious object is an IED or not. When the *human identification* goal is assigned to an agent, the agent is able to obtain information about the suspicious object from the goal. When the human peer decides that the suspicious object is an IED, the *defuse* event occurs, which triggers the *defuse IED* goal. Otherwise, the suspicious object is classified as inert. In either case, the *user interaction* role is informed

of the decision through the *inform* protocol.

The objective of the *defuser* role is to safely dispose of an IED. Disposing an IED can be done in a number of ways such as disarming the IED on the spot or moving the IED to a safer location for disarming or detonation. Agents assigned to the *defuse IED* goal can obtain the location of the IED the goal. When the *defuser* role is complete, the *user interaction* role is informed of the successful disposal of an IED through the *inform* protocol.

The behavior of the six roles are implemented by *plans* [40] that determine how agents play a given role. Furthermore, *plans* capture the domain dependant portions of the IED detection system. Each role is implemented with one plan.

5.2.1.3 Capability Model

Capabilities are a fundamental building block of any OMACS-based system. Capabilities are defined in terms of their *actions*, which are specific interactions with the environment such as retrieving the readings from a sonar, instructing a robotic arm to move, and instructing a gripper to release [27, 40]. Figure 5.9 shows the capabilities that are defined for the IED detection system.

The *user interface* capability captures the interaction mechanism between the system and a human supervisor. Currently, the *user interface* capability provides a limited display indicating the monitoring area, the location of agents, and the location of suspicious objects and their classification. In addition, the monitoring area is given by the human supervisor to the system through this capability.

The algorithm that partitions the monitor area into smaller search areas is provided by the *area division algorithm* capability. This capability is used by agents that are playing the *area divider* role.

The ability to communicate is essential for an OMACS-based system as it is used in various organizational aspects of the system such as communicating assignments and events.

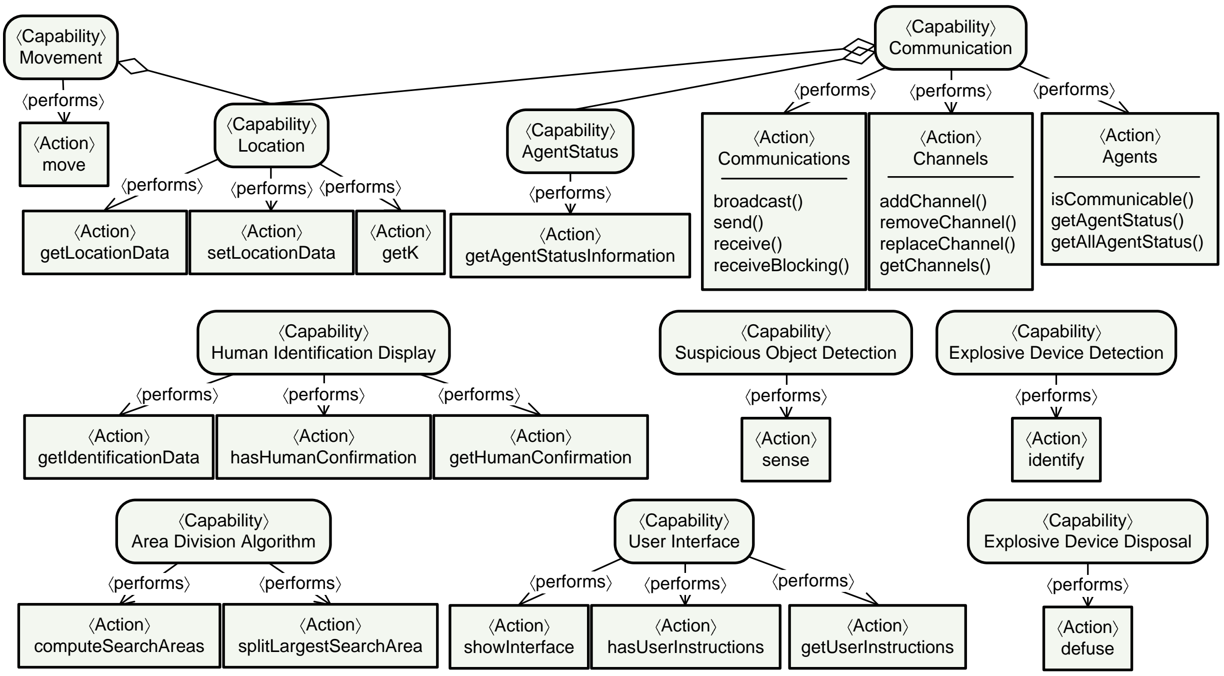


Figure 5.9: IED Capability Model

Currently, the only direct use of the *communication* capability in the system is to pass information to the *user interaction* role through the *inform* protocol.

Another important capability for any physical system is the *movement* capability. The *movement* capability provides the ability for agents to move from one location to another while providing features such as collision prevention and simple obstacle avoidance. Certain features of navigation such as path finding and plotting are too complex to be included as an action in the *movement* capability because these features could require an infinite state space.

The ability to detect objects that meet a certain classification profile (i.e., possibly containing explosive ordnance) are captured by the *suspicious object detection* capability. This capability can utilize a number of different hardware sensors such as the sonar, the camera, and some type of explosive detector.

The *explosive device detection* capability is similar to the *suspicious object detection* capability except that emphasis is on accuracy in its analysis; potential IEDs are carefully

analyzed to provide an accurate classification.

The *human identification display* capability captures another aspect of the interaction between a human peer and the system. This capability is used to present information obtained from the *explosive device detection* capability to the human peer. In addition, the response from the human peer is passed to the system through this capability.

The ability for agents to dispose IEDs is captured by the *explosive device disposal* capability. Currently, this capability utilizes a gripper to move the IED to a safe location.

5.2.1.4 Agent Model

OMACS-based systems define the types of agents by the capabilities that they possess. Figure 5.10 shows the agent model for the types that are used in the IED detection system; other agent type combinations are possible but only four are defined. Three of the agent types are defined as a hierarchy due the physical configurations of the research robots. All robots have a camera, which are used differently by the *suspicious object detection* capability and the *explosive device detection* capability, and only one robot has a gripper, which is used by the *explosive device disposal* capability.

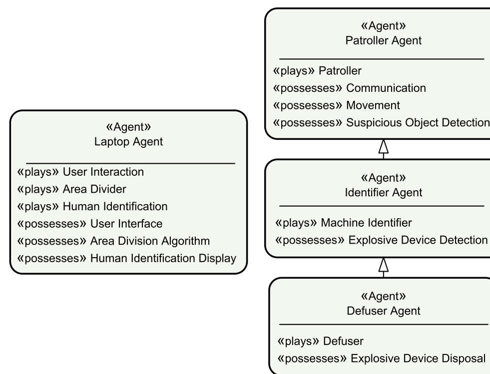


Figure 5.10: IED Agent Model

5.2.1.5 Organization-Based Agent Architecture (OBAA)

There are many possible approaches to implementing the IAL. By following the OBAA, there are numerous CCs (one CC per agent), but the CCs all work together to form the IAL. So, to a human supervisor, it appears as if there is only one CC. For simplicity, a centralized approach is used to implement the CCs to avoid additional complexities associated with adopting a distributed approach such as data synchronization. The centralized approach has two types of CC: "master" and "slave". There is one "master" CC and the rest are "slave" CC. The "master" handles all the logic processing such as handing out assignments, updating the OM with updated information, and processing events as they occur. The "slaves" on the other hand are only responsible for relaying all information to the "master" for processing.

The RA is a modified variant of the greedy task allocation algorithm, with the policy of selecting the the agent with the least workload (i.e., the agent with the least number of assignments) if multiple agents can be chosen for a specific assignment. The modified algorithm still performs "good enough" for the purpose of evaluation.

Each role defined in Figure 5.8 has a fixed priority. The RCC uses a modified version of the rate-monotonic scheduling algorithm [55], where priorities are predetermined instead of computed at runtime. The priorities for the six roles are as follows: *defuser* (highest) → *machine identifier* → *human identifier* → *user interaction* → *area divider* → *patroller* (lowest).

5.2.2 Results

The adaptive behavior of the IED detection system comes from the CCs across all agents (IAL); particularly the ORC for dealing with agent failures, which uses the RA for reorganization. The GR contains the goal model (Figure 5.7) for dealing with events such

as the *identify* and *defuse* event. In this section, the system is evaluated on its ability to adapt to agent failures, which is either the agent itself has failed or its capabilities have degraded to a point where it is unable to achieve its goal.

5.2.2.1 Robots

The IED detection system used a team of three Pioneer 3-AT (P3-AT[1]) robots ("Patroller 1 . . . 3") and one laptop ("Laptop 1"). The three robots were of the "Patroller Agent" type (Figure 5.10). The laptop was of the "Laptop Agent" type. In the experiments, after the robots have been assigned their search area(s), one of the robots would be disabled in one of two ways.

The first way was to simply just turn the robot off. The *communication* capability provided a mechanism to detect if an agent was no longer part of the network. And, for the experiments, when an agent was disconnected from the network, the agent was considered to have left the organization. When an agent left the organization, a reorganization occurred. The two remaining robots would be assigned by the ORC to take on the assignment(s) of the disabled agent.

The second way was to simulate a capability degradation by modifying the possesses score of a required capability. One of two cases can occur: either (1) the agent itself detected that it was unable to continue working on a goal and reported a failure or (2) the ORC detected that the affected agent was unable to continue working on a goal. In either case, a reorganization occurred and the assignment(s) of the affected agent would be reassigned to the two remaining robots.

Figure 5.11 shows an image of the scenario as well as the five areas (A, B, C, D, E). Eight field experiments were conducted and in each experiment, up to two robots would be disabled and the ORC would reassign the tasks from disabled robots to the remaining

[1]Further information can be obtained from http://www.activrobots.com/ROBOTS/p2at.html.

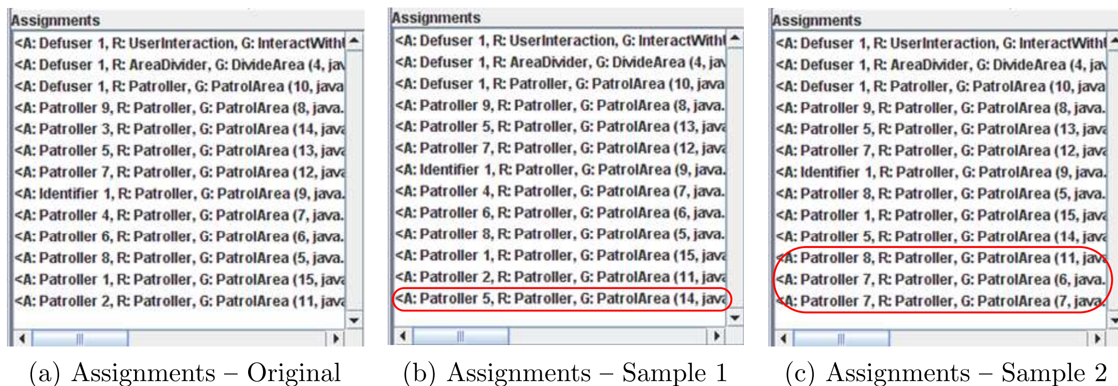
one. In one experiment, "Patroller 1" was tasked to patrol area A and B, "Patroller 2" was tasked to patrol area E, and "Patroller 3" was tasked to patrol area C and D. When "Patroller 1" was disabled, "Patroller 2" was given an additional task of patrolling area A, while "Patroller 3" was given area B. Upon disabling "Patroller 2", "Patroller 3" was tasked to patrol all five areas (A, B, C, D, and E).

Figure 5.11: IED Detection System

5.2.2.2 Simulation

To validate the results for scalability, a simulation of the IED detection system was developed on the Cooperative Robotics Organization Simulator (CROS) [115]. CROS is a high-level simulation framework for developing and testing OMACS-based systems. The test configuration consisted of eleven agents: nine agents were of the "Patroller Agent" type ("Patroller 1 . . . 9"), one was of the "Identifier Agent" type ("Identifier 1"), and one was of both the "Defuser Agent" and "Laptop Agent" type ("Defuser 1"). With a larger number of agents, more permutations were tested. Figure 5.12 illustrates just one particular sample from the simulation of how the system adapted to agent failures. Figure 5.12a shows the assignments before any agent failures occurred. There were a total of thirteen assignments; eleven of the assignments were for patrolling an area and all eleven agents were given an area to patrol because all of them were able to play the *patroller* role. For instance, the assignment

⟨A: Defuser 1, R: AreaDivider, G: DivideArea(. . .)⟩ (second assignment from the top) means that the "Defuser 1" agent is playing the *area divider* role to achieve the *divide area* goal. Figure 5.12b illustrates the change to the assignments when one agent ("Patroller 3") failed. Once the "Patroller 3" agent failed, the assignment of the failed agent was reassigned to another agent. The ORC detected the failed agent and requested from the RA the next most suitable agent ("Patroller 5") to take over. The most suitable agent was not necessarily always the same. For this particular type of assignments, there was an additional criterion of using the distance to the search area. Figure 5.12c illustrates the change to the assignments when an additional three agents ("Patroller 2", "Patroller 4", and "Patroller 6") failed. Again, the ORC detected the failed agents and new assignments were made. In this case, the assignment for "Patroller 2" is given to "Patroller 8" while "Patroller 7" takes on the assignments from "Patroller 4", and "Patroller 6".



(a) Assignments – Original (b) Assignments – Sample 1 (c) Assignments – Sample 2

Figure 5.12: Reassignments Example

Many combinations and permutations were tested in the simulation, and in all cases, the ORC was able to adapt to the failures except for cases when "Defuser 1" was one of the agent that failed. The reason for this was because the "master" CC resided in "Defuser 1" for all of the tests.

The adaptive behavior was a result of reorganization that was made possible through the use of OMACS. The assignment algorithm used one application specific policy: the

distance of the agent to the objective (either the search area, suspicious object, or IED). This policy prevents the assignment algorithm from being usable in any domain but is still usable in a wide array of domains that use mobile robots.

The experiments showed the usefulness of IAL in providing autonomous task allocation, the next section (Section 5.2.3) describes limitations of the two models (OMACS/CzM and GMoDS) with respect to organization control.

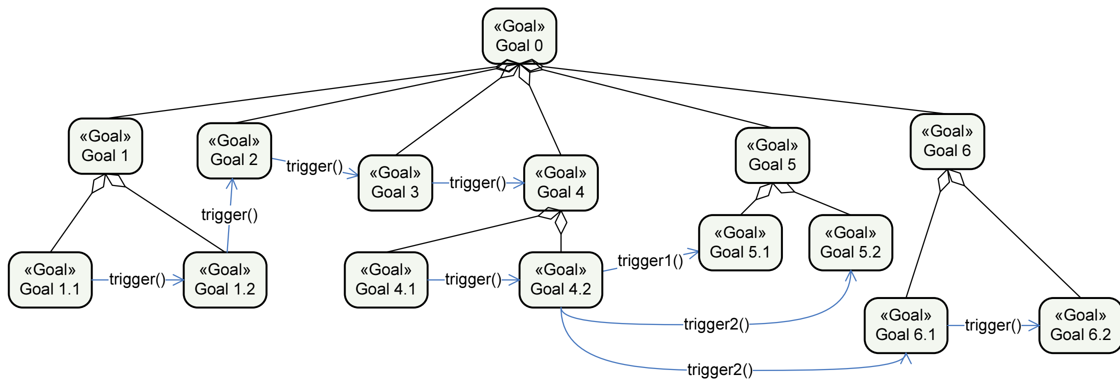
5.2.3 Organization Control Demonstration

Currently, OMACS/CzM supports two of interactions defined for organization control in Section 5.1: (1) create assignments and (2) remove assignments. However, OMACS/CzM does not explicitly support three of the interactions for roles: (1) create roles, (2) remove roles, and (3) modify roles. The remaining three interactions depend on GMoDS: (1) create goals, (2) remove goals, and (3) modify goals.

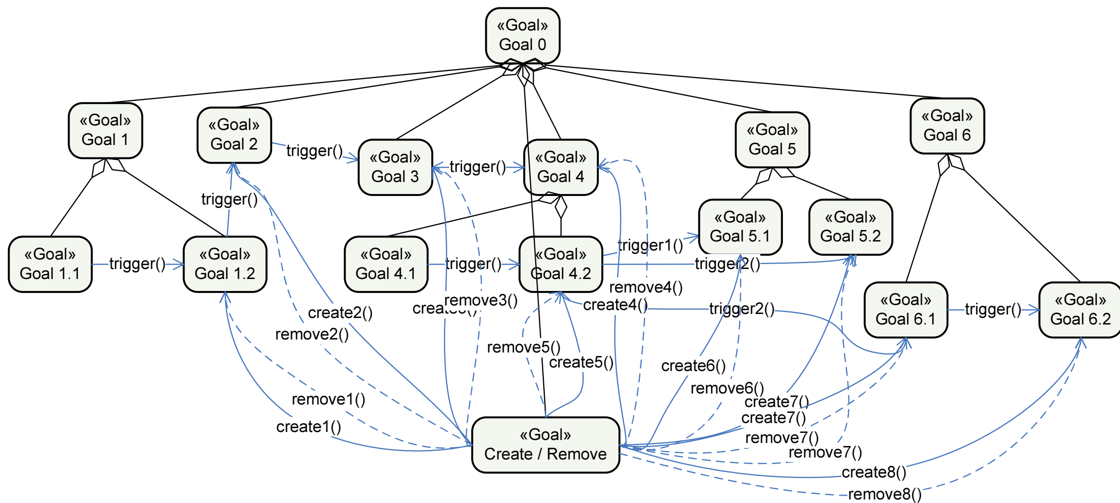
Unfortunately, the goal model (GMoDS) does not explicitly support any of interactions defined for organization control in Section 5.1. Although it is possible to include the create goals and remove goals interactions by designing them into GMoDS models, this approach is not desirable for a number of reasons.

First, this approach of designing the create and remove interactions into GMoDS model can create unnecessary clutter in the GMoDS model that may make it hard to maintain human-readability. The clutter can happen because a new goal has to be created to represent the organization control and for every goal that is desirable to allow a human supervisor to create and/or remove, a positive trigger (create) and/or a negative trigger (remove) to that goal has to be specified. All these new triggers originate from the new goal that represents organization control. Figure 5.13a shows an example GMoDS model that does not include organization control. There are nine goals that are triggerable, so these are the goals that a human supervisor is allowed to create and/or remove. Figure 5.13b shows

the modified GMoDS model where these nine goals can be created or removed by a human supervisor. There is a new goal "create / remove" that have nine positive triggers and nine negative triggers to the nine triggerable goals. Another example is the scenario described in Section 5.2.1, where the GMoDS model allows a human supervisor the ability to create and/or remove one goal even though there are five triggerable goals. Even though the clutter has negligible impact on runtime performance, the performance of validation tools would degrade significantly due to the exponential increase in the number of states.



(a) Unmodified Model

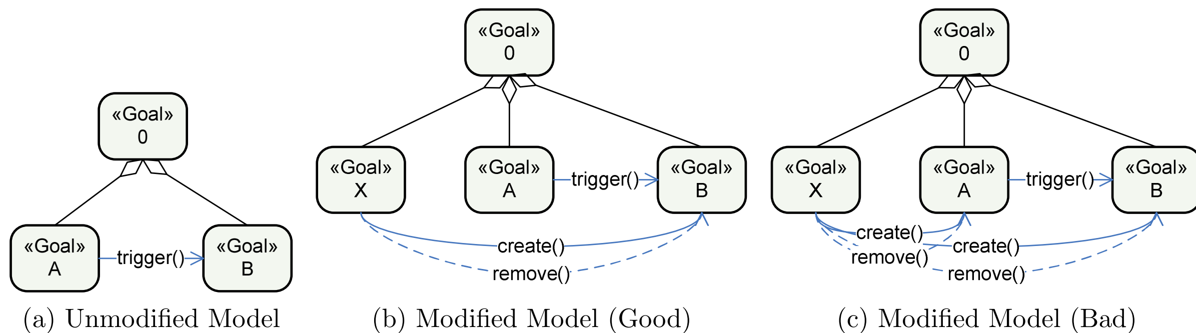


(b) Modified Model

Figure 5.13: Designing Control

Second, due to the semantics of GMoDS, it is advisable to only create triggers to goals

that are already triggerable. Doing otherwise would change the meaning of the original GMoDS model because a goal that was originally not triggerable is now triggerable. For illustration purposes, Figure 5.14a shows a very simple goal model where goal A triggers goal B. At the initial state[2] of the instance model, there are two goals present: $[0, A]$ (or $0A$, for short). For subsequent states there can be zero or more instances of goal B. For example, a trace[3] to a final state[4] can be $[0, A, B, B]$ (or $0ABB$, for short). To generalize to a regular expression, all possible traces to final states can be generalized to $0AB^{*}$[5]. Figure 5.14b shows the addition of goal X that represents the ability to create and/or remove goals by a human supervisor and there is a positive trigger and a negative trigger to goal B. At the initial state, there are three goals: $[0, X, A]$ (or $0XA$, for short). Again, all possible traces to final states can be generalized to $0XAB^{*}$. However, in Figure 5.14c, there is also a positive trigger and a negative trigger from goal X to goal A. All the possible traces to final states can be generalized to $0X(A(A|B)^{*})^{?}$[6].



(a) Unmodified Model (b) Modified Model (Good) (c) Modified Model (Bad)

Figure 5.14: Designing Control

Third, continuing with the example models from Figure 5.14b, the newly created goal X must be defined as a special type of goal. The reason for this is in the original model (Figure 5.14a), where goal B can only be achieved if goal A is achieved because that means

[2] An initial state is.

[3] A trace is a sequence of goals that are added to the instance model.

[4] A final state is where no more goals can be added to the instance model.

[5] In regular expressions, * means zero or more.

[6] In regular expressions, $^{?}$ means zero or one.

that no more instances of goal B can be triggered. So, when goal A is achieved and all instances of goal B are achieved then goal 0 is achieved. However, in Figure 5.14b, if goal X is not special, then goal B can only be achieved if both goal A and goal X is achieved. To maintain the same achievement requirement for goal B in the original model, goal X must not be part of the achievement requirement for goal B.

Fourth, because a new goal must be created in the GMoDS model to represent the ability to create and/or remove goals by a human supervisor, a role must be defined to achieve that new goal. This also requires an agent capable of playing that role to be defined. Thus, a separate mechanism must be designed and implemented to achieve this control. While there are two mechanisms for achieving this control (through roles and the IAL), it is still possible to provide the human supervisor with the illusion that there is a single mechanism by hiding the two mechanisms behind a single user interface. But that defeats the purpose behind organization control.

Fifth, the ability to create and/or remove goals by a human supervisor depends on the design of the GMoDS model and generally differs from design to design. This inconsistency results in an ambiguous situation where a human supervisor is unsure whether a triggerable goal can be created or removed. An example of this problem can be seen in the goal model (Figure 5.7) in Section 5.2.1, where a human supervisor can only create/remove one out of five triggerable goals. This problem is even more vexing since it only offers two types of organization control: creation and removal of goals.

Sixth, this approach does not facilitate the third interaction with goals: modifying goals. In GMoDS, a goal modification means changing the parameters of a goal.

The above six reasons are the limitations of the designing control through the GMoDS models, the next section (Section 5.3) explores an existing control over assignments as well as an extension to GMoDS to allow goal modification.

5.3 Organization Control

This section explores three of the interactions for organization control: creation and removal of assignments and goal modification. Creation and removal of assignments (assignment set manipulation) is already available through OMACS/CzM. Assignment set manipulation is used extensively by the CC to adapt to failures. However, assignment set manipulation has not been explicitly tested for use by a human supervisor. Section 5.3.1 explores and demonstrates the feasibility that the existing mechanism can be used by a human supervisor for manipulating assignments. As for goal modification, which currently cannot be done by GMoDS, an extension of GMoDS to allow goal modification is defined in Section 5.3.2. Furthermore, Section 5.3.2 explores and demonstrates its use by a human supervisor.

5.3.1 Assignment Set Manipulation

This section describes the procedures for manipulating assignments. OMACS/CzM already has the necessary structures to support assignment set manipulation (creation and removal of assignments). However, assignment set manipulation requires more than just modifying the data structures in OMACS/CzM. Fortunately, there are already existing mechanisms implemented that allow assignments to be created and removed because the ORC handles failures through reassignment. The existing mechanism can also be used for assignment set manipulation by a human supervisor. Even though assignments are already being manipulated by the system, it is still important for a human supervisor to be able to manipulate assignments because there are times when the human supervisor can see a better set of assignments. There are a multitude of reasons why a human supervisor can see a better set of assignments. One of the reasons could be that there is some information that is not being tracked but is affecting the performance of the system. It is unreasonable and unrealistic to expect a system to track everything. For example, perhaps the traction

of the floor is causing one of the robots to move twice as slow and in turn, that robot is taking twice as long to complete its assignment. A human supervisor can easily gain insight on this non-tracked information and tweak the assignments accordingly.

In general, when creating an assignment two steps must be performed: (1) the assignment needs to be verified for correctness and (2) the affected agent needs to be notified. As for removing an assignment, the affected agent needs to be notified.

All assignments in OMACS/CzM must conform to the validity requirements specified by OMACS/CzM. In OMACS, an assignment $\langle a, r, g \rangle$ is valid if the `rcf` is greater than 0 and the role r can achieve the goal g. In CzM, an assignment is valid if the `goodness` is greater than 0. Thus, newly created assignments must pass the validity check. An assignment that is removed does not need to be checked but it should be noted that there is now a goal that is not in the process of being achieved. Eventually, something needs to be done, either by the human supervisor or the IAL.

Once the assignment passes the validity check, the affected agent needs to be notified. Currently, there are already mechanisms for informing agents of new assignments or to stop agents from working on an existing assignment. In the CC, when a reorganization occurs autonomously (i.e., the RA produces a set of assignments), all affected agents are notified as to whether they have new assignments or to stop working on existing assignments; the CC notifies the EC of the changes. It is through the same mechanisms that affected agents can be notified when assignments are created or removed.

Allowing assignment set manipulation by a human supervisor is straightforward as most of the necessary mechanisms already exists. However, there still remains the need for an user interface for use by the human supervisor, which is beyond the scope of this dissertation. The next section (Section 5.3.1.1) demonstrates a proof-of-concept of the usefulness of assignments manipulation with a simple search scenario on how a human supervisor can manipulate assignments.

5.3.1.1 Demonstration

This section describes a scenario to illustrate a proof-of-concept of the usefulness of assignments manipulation. The scenario is a simple search scenario. There are thirty rooms that needs to be searched. The rooms are laid out along a corridor with fifteen rooms on each side. For the purpose of evaluation, all rooms are of the same size so that the time required to search a room is the same. In the scenario, there are three agents and they start at one end of the corridor. An illustration of the initial layout is depicted in Figure 5.15. The numbers that are superimposed over the rooms indicate the initial assignments, where 1 means that the room is assigned to "agent 1", 2 means that the room is assigned to "agent 2", and 3 means that the room is assigned to "agent 3".

One important criteria of the search scenario is the time taken to search all thirty rooms; the shorter the time, the better. However, due to unknown reasons, one of the agents moves twice as slow as the other two agents. As the system does not know about this problem, all three agents are considered equivalent and the system assigns ten rooms to each agent to be searched.

Without human intervention, the system would take 2590 turns to complete the search of the thirty rooms. Two agents would complete the search for ten rooms but one agent would still have five more rooms to complete. At that point, the two agents would stay idle and wait for the slow agent to complete searching the five rooms. Ideally, the two agents should search twelve rooms while the slow agent should search six rooms. This would result in a lower number of turns while keeping the time the agents complete their search to be approximately the same. However, it is not feasible to design systems that can keep track of everything or know about every potential problems and be able to deal with the problems. That is the point where a human supervisor can help alleviate the problem.

A human supervisor can notice the problem that one of the robots is moving slower than the other two robots. There are numerous ways that this can be accomplish such as a visual

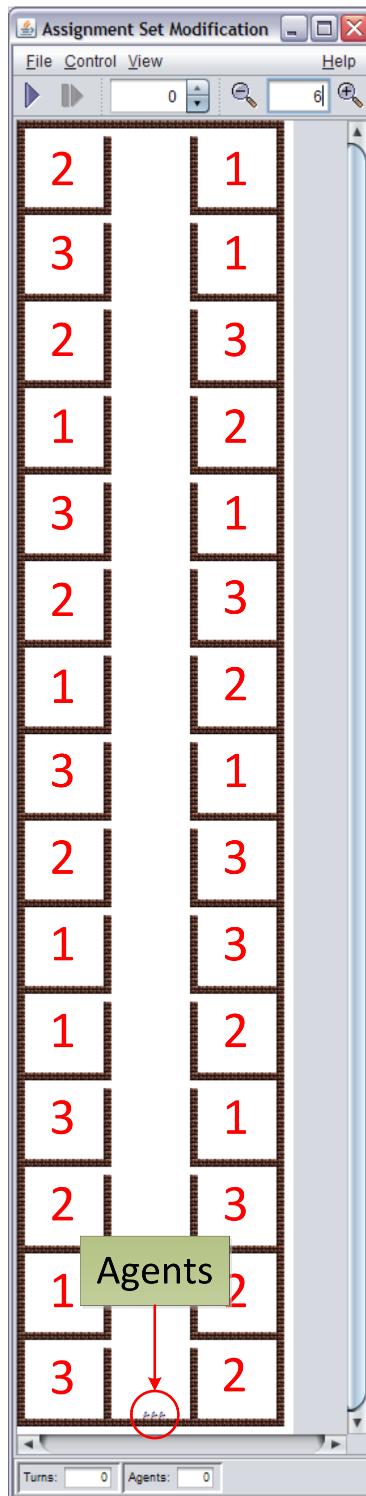


Figure 5.15: Scenario Layout

overview of the areas that have been search, keeping tabs on the search progress of each agent, keeping track of completion rate of tasks, etc. In the proof-of-concept demonstration, the assignments completed by each agent is monitored. Each room is represented by a goal, so there are thirty search goals. These goals are then assigned to the three agents. When an agent completes the search for of a room, the associated goal is achieved. The monitoring interface tracks the achievement of the search goals. Figure 5.16 shows the monitoring interface at the initial state, where each agent is assigned to search ten rooms.

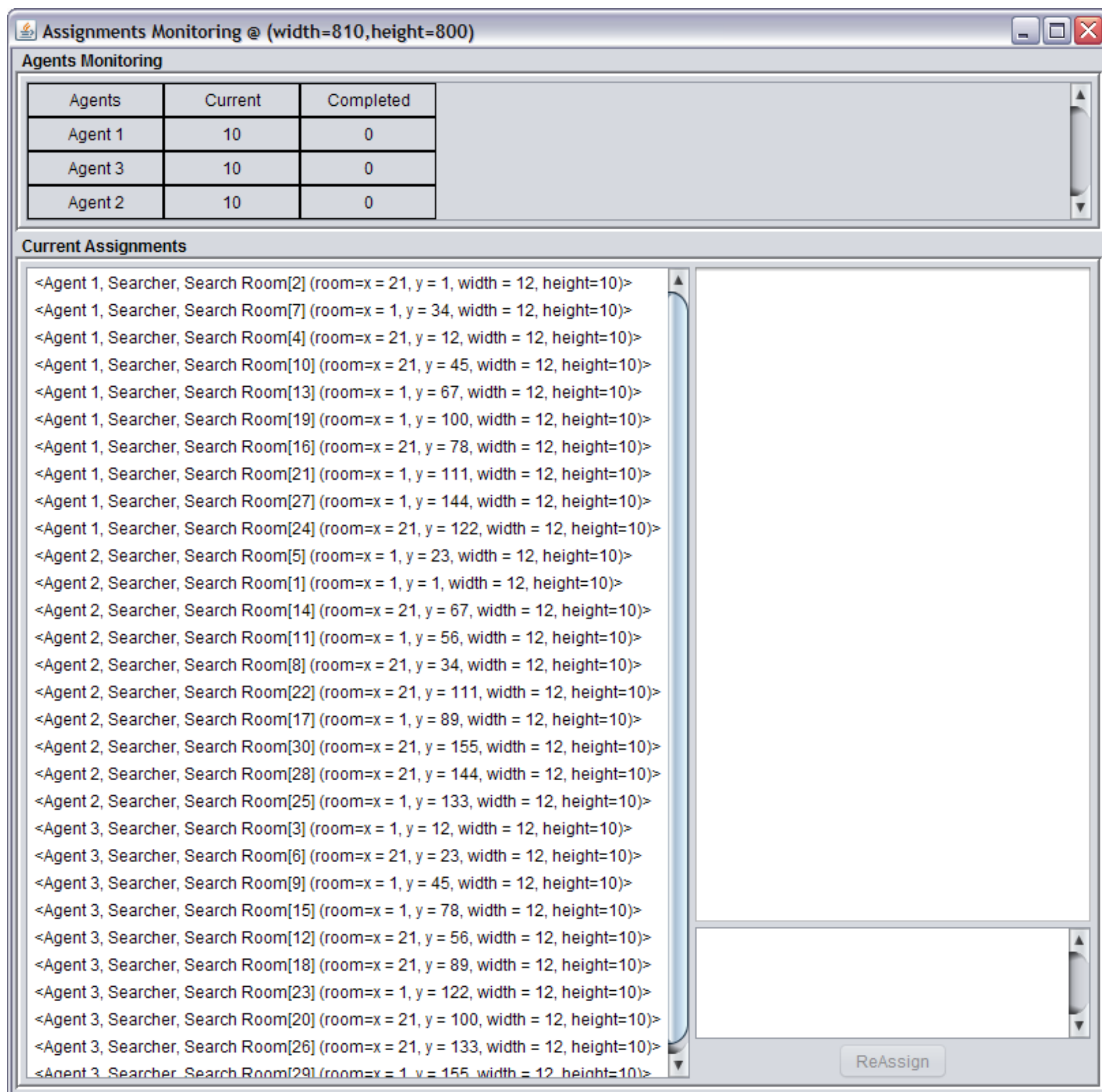
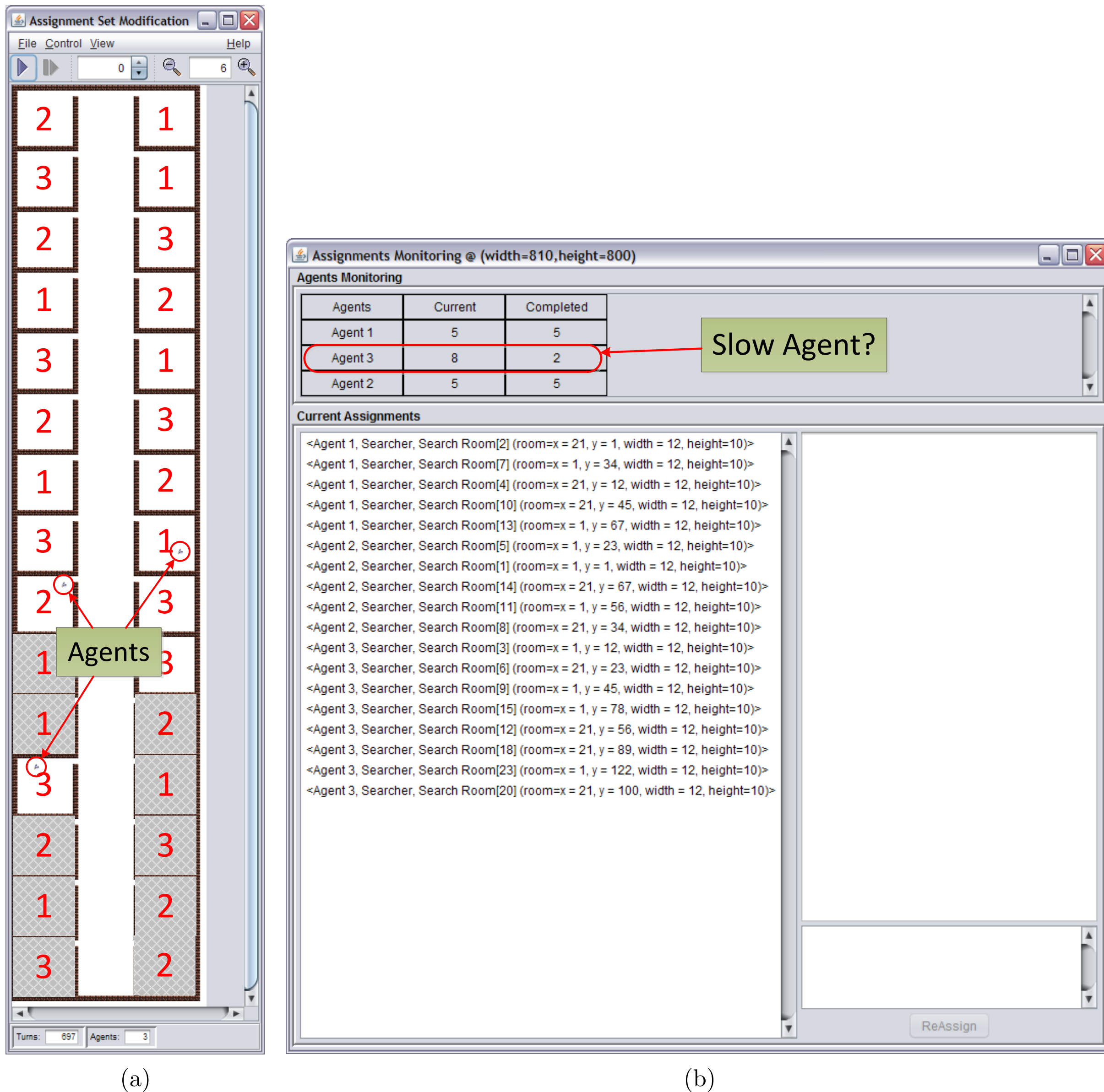


Figure 5.16: Initial State

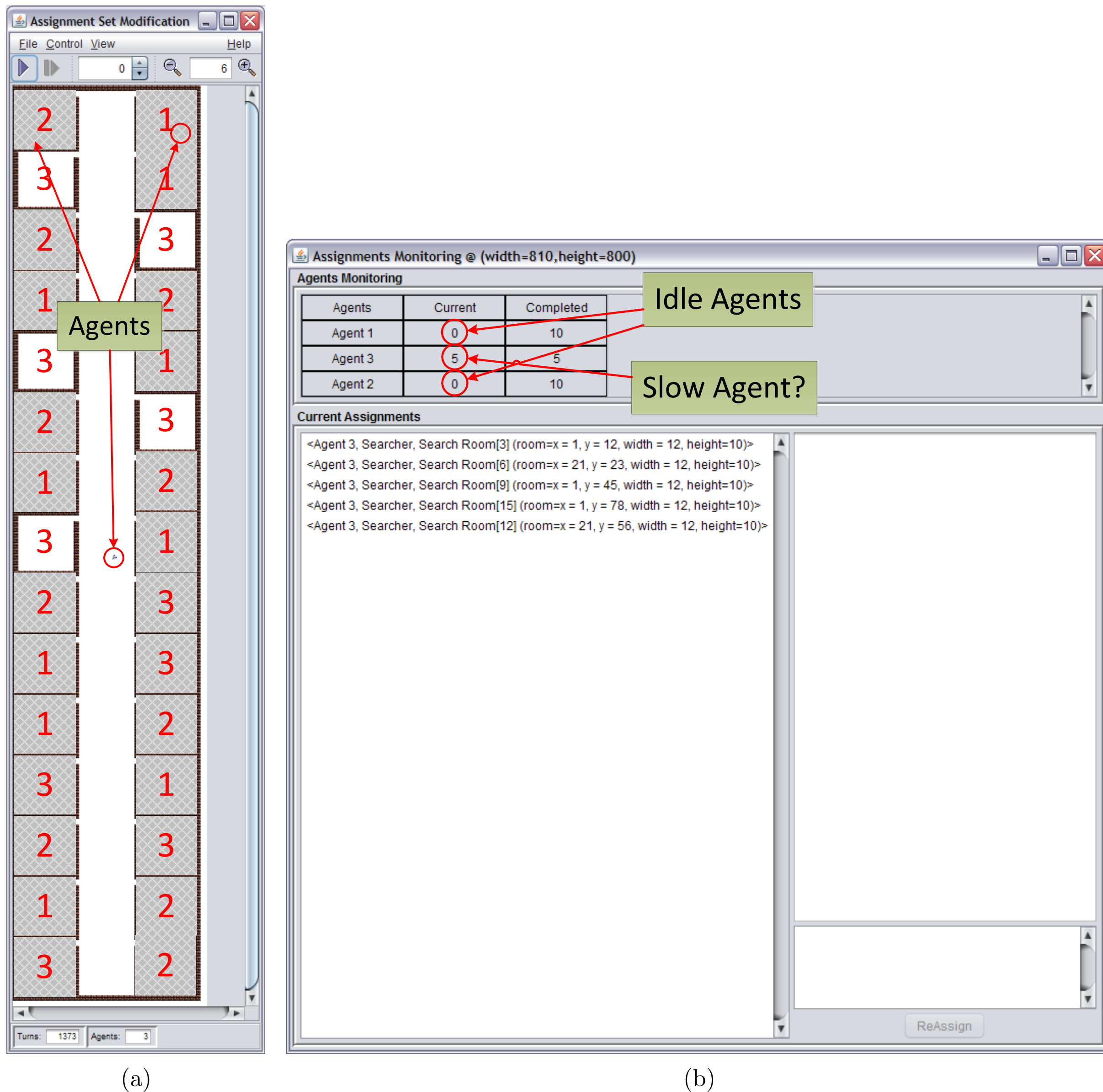
Figure 5.17 shows the point where the two normal agents have completed searching five rooms with five more rooms to complete. It is clear at this point that "agent 3" is the slow agent, which has only managed to search two rooms with eight more rooms to complete (as can be seen in Figure 5.17b). Figure 5.17a shows the locations of the three agents at the same point, where the slow agent is further apart from the other two agents.



(a) (b)

Figure 5.17: Monitoring Interface – Halfway Point

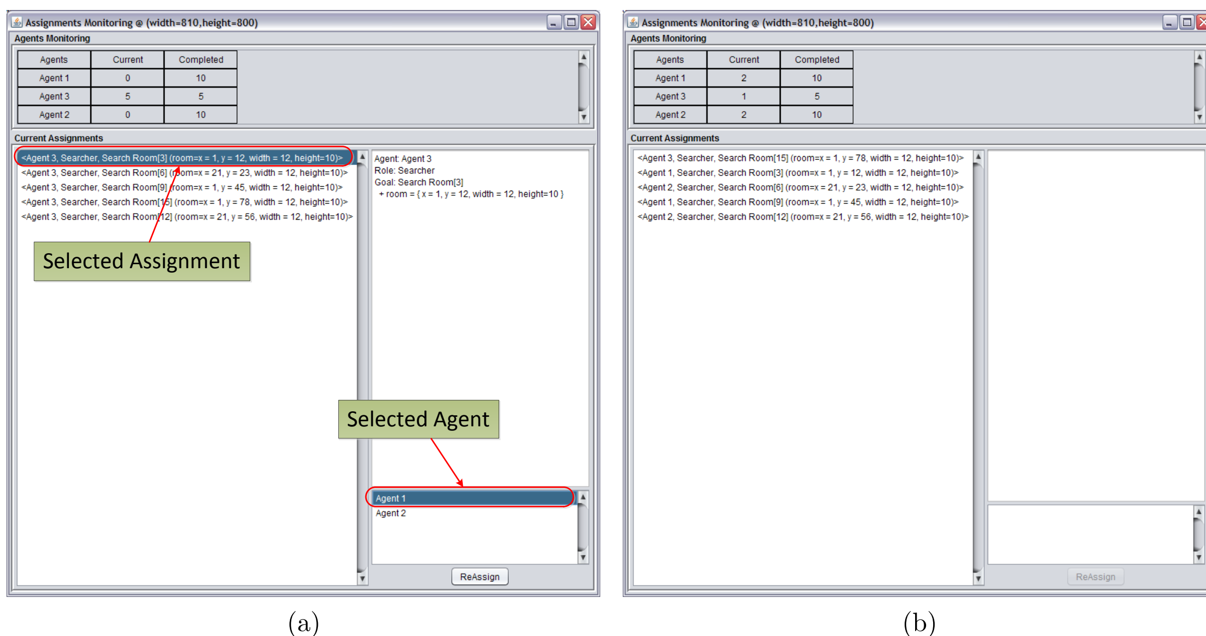
Figure 5.18 shows the point where the two normal agents have completed searching all their ten assigned rooms and are waiting for the slow agent to complete the remaining five rooms (as can be seen in Figure 5.18b). Figure 5.18a shows the locations of the three agents about halfway through the simulation. However, the slow agent has only completed two rooms.



(a) (b)

Figure 5.18: Monitoring Interface – Waiting

Even at the point where the two normal agents are just waiting for the slow agent to finish, it is still not too late for a human supervisor to manipulate the assignments. Figure 5.19 shows the reassignment process. The human supervisor selects one of the assignments and then select an agent to reassign (as can be seen in Figure 5.19a). The human supervisor repeats the process three more times such that the slow agent is left with one room to search and the other two agents are given two rooms each to search. Figure 5.19b shows the point after the reassignments are done.



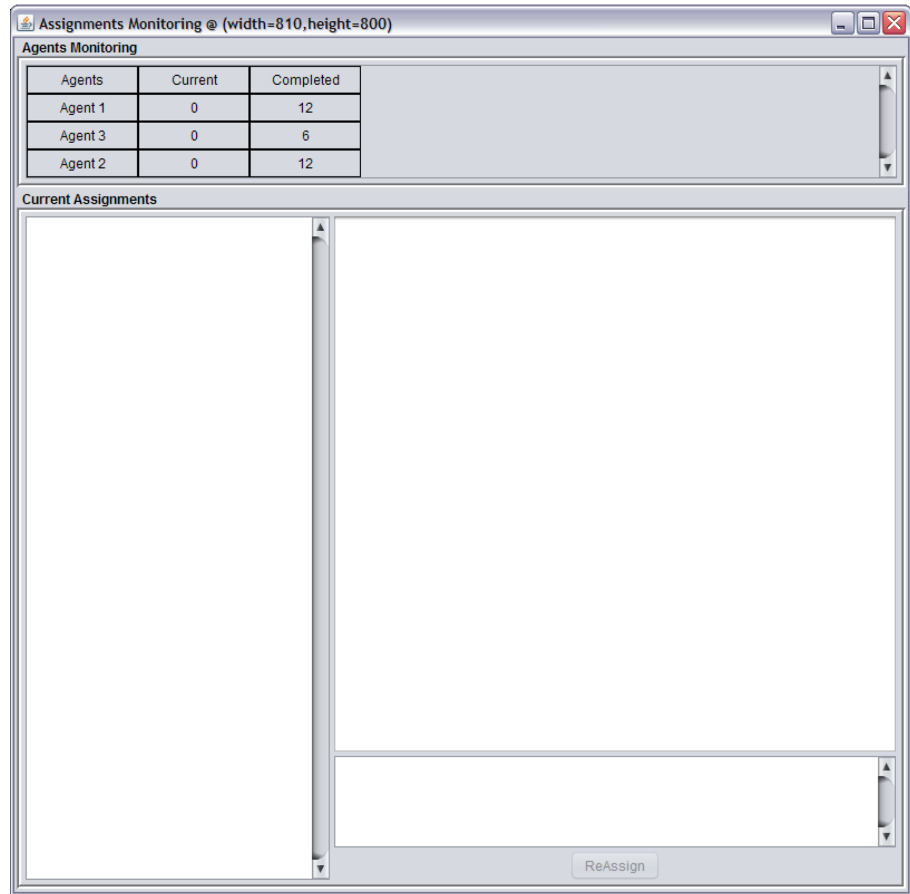
(a) (b)

Figure 5.19: Monitoring Interface – Reassignment

Figure 5.20 shows the point where the thirty rooms have been searched, where each of the two normal agents searched twelve rooms while the slow agent searched six rooms (as can be seen in Figure 5.20b, which shows the location of the agents at the point where the search is completed after the human supervisor manipulated the assignments). Furthermore, the completion time of this particular run is 1681 turns, which is 909 turns less than the 2590 turns where human intervention is not involved.

The proof-of-concept demonstration shows the usefulness of enabling a human supervisor

(a)



(b)

Figure 5.20: Monitoring Interface – Completion

to manipulate assignments. In the next section (Section 5.3.2), the interaction for manipulating goals is explored.

5.3.2 Goal Modification

This section explores goal modification. Goal modification is the only goal-related organization control that cannot be effectively "simulated" by GMoDS models as described in Section 5.2.3. GMoDS does not explicitly support goal modification although GMoDS has the necessary data structures. In GMoDS, a goal modification refers to the ability to change the values of parameters of an instance goal.

For example, the goal *survey A1* (from Chapter 2) comes from the base goal type *survey*. The base goal *survey* defines one parameter: the search area. So *survey A1* means that the search area parameter has the value of $A1$. Currently, to change the value from $A1$ to $B1$ in GMoDS, the goal *survey A1* needs to be removed (through a negative trigger) and a new goal created with $B1$ as the parameter (through a positive trigger). This process often disrupts the flow of operations because the agent assigned to *survey A1* will have to be reassigned to *survey B1*. This is not always a desirable situation. A more seamless process is needed to change values of parameters without causing a reassignment to occur.

Another example of goal modification is to modify parameters for a group of goals. For example, suppose that the *survey* goals have a parameter to indicate their search area. If the search area changes (e.g., the search area shrinks or expands), the commander wants to notify all members of a team of the change in the search area. Currently, in GMoDS, the "best" way for this notification to occur is for the commander to individually reassign each member of that team. However, a better approach would be to allow the commander to make one change to the search area for the team and all members of that team would be notified of the change automatically. In GMoDS, this can be done by exploiting the tree-based structure of goals. Non-leaf goals can be considered as team goals. If the subgoals inherit the values of their parameters from their parent goals, then the value of the parameter of the parent goal can be modified and the values of the subgoals can be automatically updated. The next part of this section describes the formalized extension to GMoDS to support goal

modification. The extension facilitates a more seamless notification mechanism as well as allowing modifications to a group of goals.

5.3.2.1 Extension

In GMoDS, all goals in the goal tree are allowed to have parameters. Simply allowing a human supervisor to change the values of parameters is not sufficient because it can break the implied relations among goals that share the same parent goal. Using the example described earlier for modifying a group of goals, it does not make sense if one member has the goal *survey A1* and another member has the goal *survey B1* and they are in the same team. A tree-based structure offers benefits that should not be ignored because that structure can be used for modifying a group of goals. However, there are no constraints that govern how these parameters interact; this result in ambiguity on how parameters should work. The extension to GMoDS introduces a number of constraints that govern how these parameters interact and function within GMoDS.

The formal specification of GMoDS is described in Section 3.2. The following is a recap of some of the definitions that will be used again. G_{S} is the set of specification goals, G_{I} is the set of instance goals, E is the set of events, T_{S} is the set of specification triggers, and T_{I} is the set of instance triggers.

In GMoDS, an instance parameter is defined as a pair $\langle k, v \rangle$, where k is the key or identifier of the parameter and v is the value of that parameter. For example, the *survey A1* goal (from Chapter 2) has one parameter: *area*, which has a value of *A1*. And so, the first two definitions is defined.

Definition 5.2. L_{key} is the set of all strings that serves as keys or identifiers. A string is a sequence of characters, where a character is an alphabet of a language.

Definition 5.3. L_{value} is the set of all strings that serves as values.

With L_{key} and L_{value} defined, parameters can be defined formally. In GMoDS, there are two models: a specification model and a runtime (also known as instance) model. In addition, goals can have multiple parameters. So, the specification model will be explained first. Since the goals in the specification model (also known as specification goals or goal types) do not need values for their parameters, a parameter for a specification goal is simply a set of $k \in L_{\text{key}}$. Suppose that a specification goal g has two parameters k_1 and k_2, then the set $\{k_1, k_2\}$ is a specification parameter group. And so, a specification parameter is defined.

Definition 5.4. A specification parameter is an element of L_{key}. A specification parameter group is a subset of L_{key}. P_{S} is the set of all specification parameter groups. A specification goal has a specification parameter group p, where $p \in P_{\text{S}}$, such that $p = \{k_1, k_2, k_3, \ldots, k_n\}$, where $k_m \in L_{\text{key}}$.

Next, is the definition of a function that returns the specification parameter group for a given specification goal.

Function 5.1. $params : G_{\text{S}} \mapsto P_{\text{S}}$. Given a specification goal g that has two parameters k_1 and k_2, then $\text{params}\,(g) = \{k_1, k_2\}$.

Similarly, a trigger (both positive and negative) can have multiple parameters. A trigger in the specification model is defined as a tuple $\langle E, G_{\text{S}}, G_{\text{S}} \rangle$. For example, if the *survey* goal type (from Chapter 2) has one parameter (*area*) and can be triggered by trigger t, then it would make sense that the set $\{area\}$ is the specification parameter group for t. And so, a specification trigger is defined.

Definition 5.5. A specification trigger has a specification parameter group p, where $p \in P_{\text{S}}$, such that $p = \{k_1, k_2, k_3, \ldots, k_n\}$, where $k_m \in L_{\text{key}}$. where tuple of $\langle T, P_{\text{S}} \rangle$. T_{S} is the set of all specification triggers.

Next, are the definitions of two functions. The first function returns the specification parameter group for a given specification trigger. The second function returns the

specification parameter groups for all specification triggers to a given specification goal. A power set is denoted by $\mathbb{P}(s)$, where s is a set.

Function 5.2. *params* : $T_{\mathrm{S}} \mapsto P_{\mathrm{S}}$. Given a specification trigger $t' = \langle t, p \rangle$, where $t = \langle e, g_1, g_2 \rangle$ and $p = \{k_1, k_2\}$, then $\mathrm{params}(t') = \{k_1, k_2\}$.

Function 5.3. *triggers* : $G_{\mathrm{S}} \mapsto \mathbb{P}(T_{\mathrm{S}})$. Given a specification goal g that can be triggered by trigger t_1 with specification parameter group $\{k_1, k_2\}$ and trigger t_2 with specification parameter group $\{k_2, k_3\}$, then $\mathrm{triggers}(g) = \{k_1, k_2, k_3\}$.

That concludes the definitions for the specification model. Next, are the definitions for the runtime (instance) model, starting with instance goals. Similar to specification goals, instance goals can also have multiple parameters. Suppose that an instance goal g has two parameters k_1 with value v_1 and k_2 with value v_2, then the set $\{\langle k_1, v_1 \rangle, \langle k_2, v_2 \rangle\}$ is an instance parameter group. And so, an instance parameter is defined.

Definition 5.6. An instance parameter is a tuple $\langle k, v \rangle$ such that $k \in L_{\mathrm{key}}$ and $v \in L_{\mathrm{value}}$. An instance parameter group is a set of the tuples $\langle L_{\mathrm{key}}, L_{\mathrm{value}} \rangle$. P_{I} is the set of all instance parameter groups. An instance goal has an instance parameter group p, where $p \in P_{\mathrm{I}}$ such that $p = \{\langle k_1, v_1 \rangle, \langle k_2, v_2 \rangle, \langle k_3, v_3 \rangle, \ldots, \langle k_n, v_n \rangle\}$, where $k_m \in L_{\mathrm{key}}$ and $v_m \in L_{\mathrm{value}}$.

Next, is the definition of a function that returns the instance parameter group for a given instance goal.

Function 5.4. *params* : $G_{\mathrm{I}} \mapsto P_{\mathrm{I}}$. Given an instance goal g that has two parameters $\langle k_1, v_1 \rangle$ and $\langle k_2, v_2 \rangle$, then $\mathrm{params}(g) = \{\langle k_1, v_1 \rangle, \langle k_2, v_2 \rangle\}$.

The following two functions are for dealing with instance parameters. The first function returns a set of $k \in L_{\mathrm{key}}$ for a given instance parameter group. The second function returns the value $v \in L_{\mathrm{value}}$ for a given instance parameter group and a key $k \in L_{\mathrm{key}}$.

Function 5.5. *keys* : $P_{\text{I}} \mapsto \mathbb{P}(L_{\text{key}})$. Given an instance parameter group p, where $p = \{\langle k_1, v_1\rangle, \langle k_2, v_2\rangle, \langle k_3, v_3\rangle\}$, then $\text{keys}(p) = \{k_1, k_2, k_3\}$.

Function 5.6. *value* : $P_{\text{I}} \times L_{\text{key}} \mapsto L_{\text{value}}$. Given an instance parameter group p, where $p = \{\langle k_1, v_1\rangle, \langle k_2, v_2\rangle, \langle k_3, v_3\rangle\}$, then $\text{value}(p, k_1) = v_1$.

Next, is the definition for instance triggers (both positive and negative). Similar to specification triggers, instance triggers can have multiple parameters. For example, if the *survey* goal type (from Chapter 2) has one parameter (*area*) and can be triggered by a trigger with the specification parameter group $\{\mathit{area}\}$. Then for the instance goal *survey A1*, it would make sense that the instance parameter group $\{\langle \mathit{area}, \mathit{A1}\rangle\}$ is the instance parameter group for the instance trigger when it occurred. This leads to the definition of an instance trigger.

Definition 5.7. An instance trigger has an instance parameter group p, where $p \in P_{\text{I}}$ such that $p = \{\langle k_1, v_1\rangle, \langle k_2, v_2\rangle, \langle k_3, v_3\rangle, \ldots, \langle k_n, v_n\rangle\}$, where $k_m \in L_{\text{key}}$ and $v_m \in L_{\text{value}}$. T_{I} is the set of all instance triggers.

The following two functions are for dealing with instance triggers. The first function returns instance parameter group for a given instance trigger. The second function returns the instance parameter group of the instance trigger that triggered the given instance goal.

Function 5.7. *params* : $T_{\text{I}} \mapsto P_{\text{I}}$. Given an instance trigger $t' = \langle t, p\rangle$, where $t = \langle e, g_1, g_2\rangle$ and $p = \{\langle k_1, v_1\rangle, \langle k_2, v_2\rangle\}$, then $\text{params}(t') = \{\langle k_1, v_1\rangle, \langle k_2, v_2\rangle\}$.

Function 5.8. *triggers* : $G_{\text{I}} \mapsto T_{\text{I}}$. Given an instance goal g that can be triggered by trigger t_1 with specification parameter group $\{k_1, k_2\}$ and trigger t_2 with specification parameter group $\{k_2, k_3\}$ and g was triggered by t_1 with instance parameter group $\{\langle k_1, v_1\rangle, \langle k_2, v_2\rangle\}$, then $\text{triggers}(g) = \{\langle k_1, v_1\rangle, \langle k_2, v_2\rangle\}$.

Now that parameters for goals and triggers have been formally defined, modifications can be defined. In GMoDS, a goal modification means to change the value of a parameter. Since an instance parameter is the tuple $\langle k, v \rangle$, then a modification is the change to the value of a parameter. For example, if the value of the *area* parameter of the instance goal *survey A1* (from Chapter 2) is to be changed to *B1*, then the modification for *survey A1* is $\{\langle area, B2 \rangle\}$. So, a modification is defined.

Definition 5.8. A modification is a tuple $\langle g \in G_{\mathrm{I}}, p \in P_{\mathrm{I}} \rangle$ such that $\forall k \in \mathrm{keys}\,(p) \Rightarrow k \in \mathrm{keys}\,(\mathrm{params}\,(g))$. M is the set of all modifications.

The following two functions are for interacting with modifications. The first function returns the instance parameter group for a given modification. The second function returns the latest modification for each instance parameter of a given instance goal.

Function 5.9. *params* : $M \mapsto P_{\mathrm{I}}$. Given a modification $m = \langle g, \{\langle k_1, v_1 \rangle, \langle k_2, v_2 \rangle\} \rangle$, then $\mathrm{params}\,(m) = \{\langle k_1, v_1 \rangle, \langle k_2, v_2 \rangle\}$.

Function 5.10. *mod* : $G_{\mathrm{I}} \mapsto M$. Given an instance goal g with three instance parameters $(\langle k_1, v_1 \rangle,\ \langle k_2, v_2' \rangle$, and $\langle k_3, v_3'' \rangle)$ and g has been modified by a sequence of modifications $[m_1, m_2, m_3] \in M$, where $m_1 = \langle g, \{\langle k_2, v_2' \rangle, \langle k_3, v_3' \rangle\} \rangle$, $m_2 = \langle g, \{\langle k_3, v_3'' \rangle\} \rangle$, and $m_3 = \langle g, \{\} \rangle$. Then $\mathrm{mod}\,(g) = \langle g, \{\langle k_2, v_2' \rangle, \langle k_3, v_3'' \rangle\} \rangle$.

The next function is for retrieving the initial values of the instance parameter group for a given instance goal.

Function 5.11. *init* : $G_{\mathrm{I}} \mapsto P_{\mathrm{I}}$. Given an instance goal g with three instance parameters $\langle k_1, v_1' \rangle$, $\langle k_2, v_2'' \rangle$, and $\langle k_3, v_3''' \rangle)$ and that v_1 is the initial value of k_1, v_2 is the initial value of k_2, and v_3 is the initial value of k_3. Then $\mathrm{init}\,(g) = \{\langle k_1, v_1 \rangle, \langle k_2, v_2 \rangle, \langle k_3, v_3 \rangle\}$.

Now that all the necessary parts have been defined, the following five constraints limits how the parameters interact. The first constraint (Constraint 5.1) specifies how parameters

interact within the specification model. All parameters of a specification goal must come from either its parent goal or defined as a parameter of a trigger to that specification goal. For instance (as shown in Figure 5.21), given (1) a specification goal g_1 that has the specification parameter group $\{k_1, k_2, k_3, k_4\}$; (2) g_1 is a subgoal of g_0; and (3) two triggers (t_1, t_2) to g_1. The first constraint says that k_m must either be a parameter of g_0 or defined as a parameter of t_1 or t_2. Thus, k_1 and k_3 are from g_0, k_2 is from t_1, and k_4 is from t_2.

$$\begin{aligned}&\forall g \in G_{\mathrm{S}}, t \in T_{\mathrm{S}}\\&\quad t \in \text{triggers}(g) \Rightarrow \text{params}(g) = (\text{params}(\text{parent}(g)) \cup \text{params}(t))\end{aligned} \tag{5.1}$$

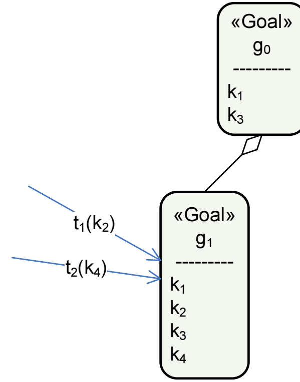
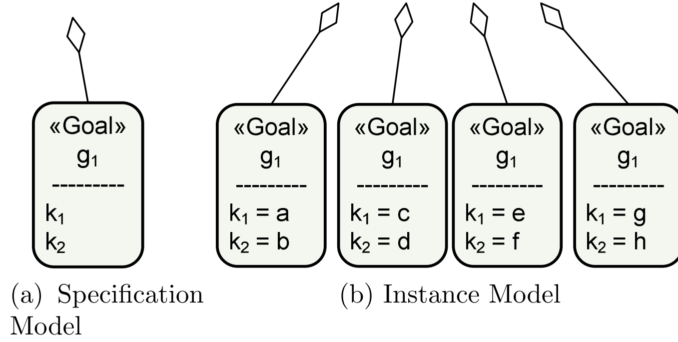


Figure 5.21: Source of Parameters

The second constraint (Constraint 5.2) specifies a relationship between the parameters of a specification goal and the parameters of instances of that specification goal. All instance goals must have the same set of keys as the associated specification goal. For instance (as shown in Figure 5.22), given a specification goal g_1 that has specification parameter group $\{k_1, k_2\}$ (shown in Figure 5.22a), then all instances of g_1 (which are instance goals) must have the instance parameter group $\{\langle k_1, x\rangle, \langle k_2, y\rangle\}$ (shown in Figure 5.22b).

$$\begin{aligned}&\forall g \in G_{\mathrm{I}}, k \in L_{\text{key}}\\&\quad k \in \text{keys}(\text{params}(g)) \Leftrightarrow k \in \text{params}(\text{spec}(g))\end{aligned} \tag{5.2}$$



(a) Specification Model

(b) Instance Model

Figure 5.22: Same Set of Keys

The third constraint (Constraint 5.3) specifies a relationship among the values of instance parameters. The value for an instance parameter must come from one of three sources: its parent goal, the trigger that created that instance goal, or a modification to that instance goal.

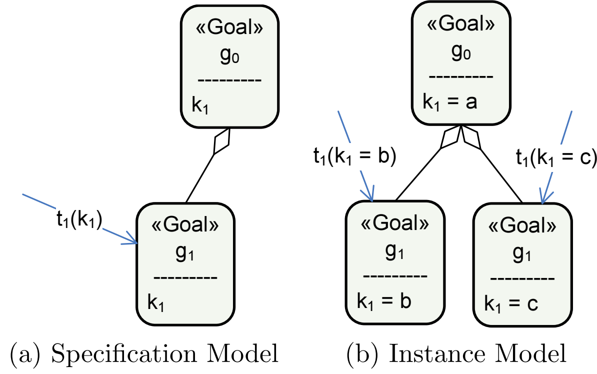
$$
\begin{aligned}
&\forall g \in G_{\mathrm{I}}, k \in L_{\text{key}} \\
&\quad k \in \operatorname{keys}(\operatorname{params}(g)) \Rightarrow \\
&\quad\quad \operatorname{value}(\operatorname{params}(g), k) = \operatorname{value}(\operatorname{params}(\operatorname{parent}(g)), k) \vee \\
&\quad\quad \operatorname{value}(\operatorname{params}(g), k) = \operatorname{value}(\operatorname{params}(\operatorname{triggers}(g)), k) \vee \\
&\quad\quad \operatorname{value}(\operatorname{params}(g), k) = \operatorname{value}(\operatorname{params}(\operatorname{mod}(g)), k)
\end{aligned}
\tag{5.3}
$$

The fourth constraint (Constraint 5.4) expands on the third constraint by limiting the source of the initial value for an instance parameter. If a parameter is defined in any of the triggers, then the value of that parameter must come from the trigger. For instance (as shown in Figure 5.23), given (1) two specification goals g_1 and g_0, (2) g_1 is a subgoal of g_0, (3) g_1 and g_0 both have the parameter k_1, and (4) there is a trigger t_1 to g_1 with parameter k_1 (shown in Figure 5.23a), then the value of parameter k_1 for all instances of g_1 must come from the trigger (shown in Figure 5.23b).

$$\forall g \in G_{\mathrm{I}}, k \in L_{\mathrm{key}}$$

$$k \in (\operatorname{keys}(\operatorname{params}(\operatorname{triggers}(g))) - \operatorname{keys}(\operatorname{params}(\operatorname{mod}(g)))) \Rightarrow \tag{5.4}$$

$$\operatorname{value}(\operatorname{params}(g), k) = \operatorname{value}(\operatorname{params}(\operatorname{triggers}(g)), k)$$



(a) Specification Model (b) Instance Model

Figure 5.23: Values are from Triggers

The fifth constraint (Constraint 5.5) limits the subsequent sources of the values of an instance parameter to be either from the parent instance goal if that instance parameter can inherited the value from the parent instance goal or from a modification to that instance goal.

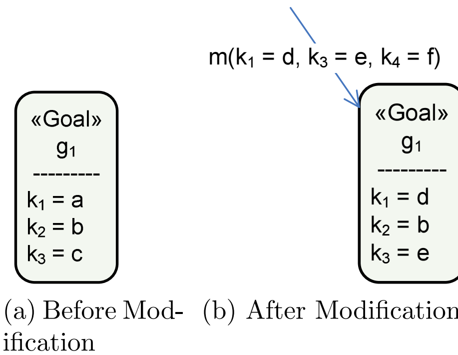
$$\forall g \in G_{\mathrm{I}}, k \in L_{\mathrm{key}}$$

$$\begin{aligned} & k \in \operatorname{keys}(\operatorname{params}(g)) \wedge \operatorname{value}(\operatorname{params}(g), k) \neq \operatorname{value}(\operatorname{init}(g), k) \Rightarrow \\ & \quad \left(\begin{aligned} & k \notin \operatorname{keys}(\operatorname{params}(\operatorname{triggers}(g))) \Rightarrow \\ & \quad \operatorname{value}(\operatorname{params}(g), k) = \operatorname{value}(\operatorname{params}(\operatorname{mod}(g)), k) \vee \\ & \quad \operatorname{value}(\operatorname{params}(g), k) = \operatorname{value}(\operatorname{params}(\operatorname{parent}(g)), k) \end{aligned} \right) \\ & \quad \wedge \\ & \quad \left(\begin{aligned} & k \in \operatorname{keys}(\operatorname{params}(\operatorname{triggers}(g))) \Rightarrow \\ & \quad \operatorname{value}(\operatorname{params}(g), k) = \operatorname{value}(\operatorname{params}(\operatorname{mod}(g)), k) \end{aligned} \right) \end{aligned} \tag{5.5}$$

The five constraints govern how parameters interact and function within GMoDS.

Next, there are two operations that describe how modifications work. The first operation (Equation 5.6) states that when a modification $\langle g, p \rangle$ occurs for instance goal g, then all values of matching parameters are changed. For instance (as shown in Figure 5.24), if instance goal g_1 has three parameters $\langle k_1, a \rangle$, $\langle k_2, b \rangle$, and $\langle k_3, c \rangle$ (shown in Figure 5.24a). There is a modification $\{\langle k_1, d \rangle, \langle k_3, e \rangle, \langle k_4, f \rangle\}$ to g_1, then the new values of the parameters for g_1 is $\{\langle k_1, d \rangle, \langle k_2, b \rangle, \langle k_3, e \rangle\}$ (shown in Figure 5.24b). Even though $\langle k_4, v_4 \rangle$ is part of the modification, nothing happens for that parameter because k_4 does not exists in the g_1; modifications do not add new parameters. The current state is denoted as S and S' denotes the next state.

$$
\begin{aligned}
& S \, [\![\text{modify}\,(g \in G_{\text{I}}, p \in P_{\text{I}})]\!] \, S' \\
& S' \models \quad \begin{aligned} & \forall k \in L_{\text{key}} \\ & \quad k \in (\text{keys}\,(p) \cap \text{keys}\,(\text{params}\,(g))) \Rightarrow \\ & \qquad \text{value}\,(\text{params}\,(g')\,, k) = \text{value}\,(p, k) \end{aligned}
\end{aligned} \tag{5.6}
$$



(a) Before Modification (b) After Modification

Figure 5.24: Goal Modification

In GMoDS, goals are decomposed into subgoals and achieving the subgoals automatically achieves the parent goal. Likewise, in a similar manner, modifications to an instance goal follow a similar principal. However, instead of moving up to the parent goal, modifications are moving down to the children goals. The second operation (Equation 5.7) states that when

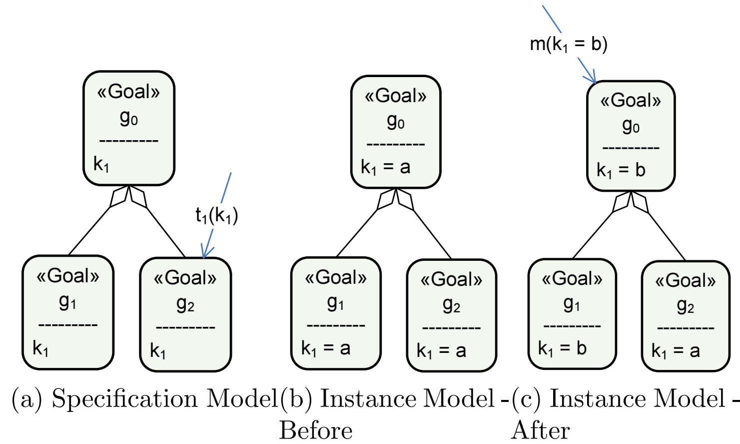
a modification occurs, all values of matching and inherited parameters for the subgoals are also changed. It is important to know that the fourth constraint (Equation 5.4) determines whether a parameter is inherited; an inherited parameter is a parameter that exists only in the parent goal. For instance (as shown in Figure 5.25), given (1) three specification goals g_0, g_1, and g_2, (2) g_1 and g_2 are subgoals of g_0, (3) g_0, g_1, and g_2 have the parameter k_1, and (4) there is a trigger t_1 to g_2 with parameter k_1 (shown in Figure 5.25a). And in the instance model, the value of the parameter k_1 of g_0, g_1, and g_2 is a (Figure 5.25b). There is a modification $\{\langle k_1, b\rangle\}$ to g_0, then the value of the parameter k_1 of g_0 and g_1 changes to b but the value of the parameter k_1 of g_2 remains unchanged (Figure 5.25c). This is because even though the parameter k_1 from g_2 is the same key as the parameter from g_0, k_1 was redefined by the trigger t_1. Thus, the parameter k_1 from g_2 is not an inherited parameter.

$$
\begin{array}{l}
S \,[\![\mathrm{modify}\left(g_1 \in G_{\mathrm{I}}, p \in P_{\mathrm{I}}\right)]\!]\, S' \\
S' \models \quad \begin{array}{l}
\forall g_2 \in G_{\mathrm{I}}, k \in L_{\mathrm{key}} \\
\quad g_2 \in \mathrm{descendant}\left(g_1\right) \wedge \\
\quad k \in \left(\begin{array}{c}\left(\begin{array}{c}\mathrm{keys}\left(p\right) \cap \mathrm{keys}\left(\mathrm{params}\left(g_2\right)\right) \cap \\ \mathrm{keys}\left(\mathrm{params}\left(\mathrm{parent}\left(g_2\right)\right)\right)\end{array}\right) - \\ \mathrm{keys}\left(\mathrm{params}\left(\mathrm{triggers}\left(g_2\right)\right)\right)\end{array}\right) \Rightarrow \\
\qquad \mathrm{value}\left(\mathrm{params}\left(g_2'\right), k\right) = \mathrm{value}\left(\mathrm{params}\left(\mathrm{parent}\left(g_1\right)\right), k\right)
\end{array}
\end{array}
\tag{5.7}
$$

This concludes the formal description of the extension to GMoDS to support goal modification. The next section (Section 5.3.2.2) looks at the proof of correctness of the extension.

5.3.2.2 Proof

This section provides a proof-by-exhaustion that goal modifications ($\mathrm{modify}\left(g \in G_{\mathrm{I}}, p \in P_{\mathrm{I}}\right)$) do not violate the three constraints as defined in Section 5.3.2.1: Constraint 5.3,



(a) Specification Model (b) Instance Model - Before (c) Instance Model - After

Figure 5.25: Propagation of Modifications

Constraint 5.4, and Constraint 5.5. In the context of this proof, Constraint 5.1 is assumed to hold because the constraint only applies to the specification tree and goal modification only applies to the instance tree. Similarly, Constraint 5.2 is also assumed to hold because the constraint states that instance goals must have the same set of parameters as their associated specification goals and goal modifications do not add or remove parameters.

Proof. The goal modification operation is defined by Equation 5.6 and Equation 5.7. There are two cases to consider when a goal modification ($\text{modify}\left(g \in G_{\text{I}}, p \in P_{\text{I}}\right)$) occurs on the given instance goal g with instance parameter group p: (1) how the modification is applied to the given instance goal g and (2) how the modification is applied to the descendant goals of g.

1. How the modification is applied to instance goal g. In Equation 5.6, the value of an instance parameter is changed when $k \in \left(\text{keys}\left(p\right) \cap \text{keys}\left(\text{params}\left(g\right)\right)\right)$ and nothing happens to the values of instance parameters that are not modified, $k \notin \left(\text{keys}\left(p\right) \cap \text{keys}\left(\text{params}\left(g\right)\right)\right)$. The instance parameters of g and instance parameters of p determines the modification. Thus, there are five possible cases to consider.

   (a) $\text{keys}\left(\text{params}\left(g\right)\right) \subset \text{keys}\left(p\right)$. Any $k \notin \text{keys}\left(\text{params}\left(g\right)\right) \wedge k \in \text{keys}\left(p\right)$

is ignored because the instance parameter with key k does not exists in instance goal g and goal modification does not add or remove instance parameters. Because $\text{keys}(\text{params}(g)) \subset \text{keys}(p)$, the values of all instance parameters of g will be modified. This results in "$\text{value}(\text{params}(g), k) = \text{value}(\text{params}(\text{mod}(g)), k)$" being true for all instance parameters of g. And so, Constraint 5.3 and Constraint 5.5 remains satisfied. Constraint 5.4 is vacuously true because every instance parameters of the instance goal g is modified, $\text{keys}(\text{params}(\text{triggers}(g))) - \text{keys}(\text{params}(\text{mod}(g))) = \emptyset$.

(b) $\text{keys}(\text{params}(g)) \supset \text{keys}(p)$. For any $k \in \text{keys}(\text{params}(g)) \wedge k \notin \text{keys}(p)$, the value associated with k remains unchanged by the goal modification because $k \notin (\text{keys}(p) \cap \text{keys}(\text{params}(g)))$. Constraint 5.3 and Constraint 5.5 remains satisfied because for instance parameters that are modified, "$\text{value}(\text{params}(g), k) = \text{value}(\text{params}(\text{mod}(g)), k)$" is true. And for instance parameters that are unchanged, the constraints are satisfied before the modification and so, continues to satisfy the constraints. Constraint 5.4 holds because the constraint is satisfied prior to the modification for all $k \in \text{keys}(\text{params}(g))$ and the remaining keys continue to hold true because $(\text{keys}(p) - \text{keys}(\text{params}(g))) \subset \text{keys}(\text{params}(g))$.

(c) $\text{keys}(\text{params}(g)) = \text{keys}(p)$. Similar to case, $\text{keys}(\text{params}(g)) \subset \text{keys}(p)$, with the exception that $\nexists k \mid k \notin \text{keys}(\text{params}(g)) \wedge k \in \text{keys}(p)$.

(d) $\text{keys}(\text{params}(g)) \cap \text{keys}(p) \neq \emptyset$. There are three cases to consider.

   i. $k \notin \text{keys}(\text{params}(g)) \wedge k \in \text{keys}(p)$. Since goal modification do not add or remove instance parameters, the extra instance parameters from the modification are ignored. So, the three constraints are vacuously true.

   ii. $k \in \text{keys}(\text{params}(g)) \wedge k \notin \text{keys}(p)$. Since Constraint 5.3, Constraint 5.5, and Constraint 5.4 are satisfied prior to the modification and none of the

instance parameters in this case are modified, the three constraints continue to be satisfied.

iii. $k \in \text{keys}(\text{params}(g)) \wedge k \in \text{keys}(p)$. Similar to case, $\text{keys}(\text{params}(g)) \supset \text{keys}(p)$, where instance parameters that are modified will still satisfy the constraints because "$\text{value}(\text{params}(g), k) = \text{value}(\text{params}(\text{mod}(g)), k)$" is true. And instance parameters that are not modified continue to satisfy the constraints because they satisfy the constraints prior to modification.

(e) $\text{keys}(\text{params}(g)) \cap \text{keys}(p) = \emptyset$. No values associated with any instance parameters of the instance goal g is modified, and since g satisfies Constraint 5.3, Constraint 5.5, and Constraint 5.4 before the modification, g continues to satisfy the three constraints.

2. How the modification is applied to the descendant goals of g. In Equation 5.7, the type of descendant goal is determined by the instance parameters of the descendant goal g', the instance parameters of the parent instance goal of g', the triggers to g', and p. For this portion of the proof, the proof evaluates g and the children as the proof can be applied recursively from the instance goal g to the leaf goals of g. So the $\text{parent}(g') = g$. Thus, there are four cases to consider.

   (a) $\text{keys}(\text{params}(g)) \cap \text{keys}(\text{params}(g')) = \emptyset$. Since the result is empty, the triggers to g' (if there are any) and p do not change the type of children goal. Since $\text{keys}(p) \cap \emptyset = \emptyset$, that means that no instance parameters of g' are modified. And since g' satisfies Constraint 5.3, Constraint 5.5, and Constraint 5.4 prior to the modification, g' continues to satisfy the three constraints.

   (b) $\text{keys}(\text{params}(g)) \cap \text{keys}(\text{params}(g')) \cap \text{keys}(p) = \emptyset$. Since the result is empty, the triggers to g' (if there are any) do not change the type of children goal. Since $\emptyset - \text{keys}(\text{params}(\text{triggers}(g'))) = \emptyset$, that means that no instance parameters of g' are

modified. And since g' satisfies Constraint 5.3, Constraint 5.5, and Constraint 5.4 prior to the modification, g' continues to satisfy the three constraints.

(c) $\left(\begin{array}{c}(\text{keys}\,(\text{params}\,(g)) \cap \text{keys}\,(\text{params}\,(g')) \cap \text{keys}\,(p)) - \\ \text{keys}\,(\text{params}\,(\text{triggers}\,(g')))\end{array}\right) = \emptyset$. Since the result is $\emptyset$, that means that no instance parameters of g' are modified. And since g' satisfies Constraint 5.3, Constraint 5.5, and Constraint 5.4 prior to the modification, g' continues to satisfy the three constraints.

(d) $\left(\begin{array}{c}(\text{keys}\,(\text{params}\,(g)) \cap \text{keys}\,(\text{params}\,(g')) \cap \text{keys}\,(p)) - \\ \text{keys}\,(\text{params}\,(\text{triggers}\,(g')))\end{array}\right) \neq \emptyset$. There are two cases to consider based on whether any of the triggers to g' contain the same keys as the instance parameters of g'.

   i. $k \in \text{keys}\,(\text{params}\,(\text{triggers}\,(g')))$. The instance parameter k will not inherit its value from the parent instance goal g. This results in a $p' \subset p \mid k \notin \text{keys}\,(\text{params}\,(p'))$.

   ii. $k \notin \text{keys}\,(\text{params}\,(\text{triggers}\,(g')))$. The instance parameter k will inherit its value from parent instance goal g. This results in a $p' \subset p \mid k \in \text{keys}\,(\text{params}\,(p'))$

   The instance parameters of the instance goal g' will be modified recursively $\text{modify}\,(g', p')$.

Thus, the goal modification operation does not violate Constraint 5.3, Constraint 5.5, and Constraint 5.4. □

This concludes the proof for the goal modification operation. The next section (Section 5.3.2.3) demonstrates a proof-of-concept of the usefulness of goal modification.

5.3.2.3 Demonstration

This section describes a navigation scenario to illustrate a proof-of-concept of the usefulness of goal modification. The navigation scenario has a team of robots that navigates along a given set of waypoints. The navigation scenario is based on a much larger reconnaissance scenario from the Human Robot Teams (HuRT)[7] project. In the navigation scenario, the team consists of four robots (simulated via Player/Stage[8]). The four robots navigate as a team using three pieces of information obtained from their goals: a set of waypoints, the danger level, and the formation. Once the robots have their goals, they begin to navigate along the given set of waypoints in the specified formation. The danger level determines the distance between robots; a high danger level means that robots are further apart while a low danger level means that robots are closer to one another.

Figure 5.26 shows a partial view of the goal model from the reconnaissance scenario. The *Go To* goal is the navigation goal for the team. One of the subgoals of the *Go To* goal is the *Move* goal, which has four subgoals. Each subgoal represents the position of a robot in a formation. As defined by the goal modification extensions (Section 5.3.2.1), the *location*, *formation*, *level*, and *points* parameters are inherited parameters. Thus, changing the values of those parameters from the *Go To* goal will change the values of the *Move*, *LeadMove*, *AlphaMove*, *BetaMove*, and *GammaMove* goals. Likewise, changing the values of those parameters from the *Move* goal will also change the values of the *LeadMove*, *AlphaMove*, *BetaMove*, and *GammaMove* goals.

In a run of the system, a human supervisor initiates the *Go To* task (which is represented by the *Go To* goal) and provides three waypoints, the medium danger level, the wedge[9] formation. As the team begins to navigate along the given waypoints, the human supervisor

[7]More information about HuRT can be obtained from http://projects.cis.ksu.edu/gf/project/hurt/.

[8]More information about Player/Stage can be obtained from http://playerstage.sourceforge.net/.

[9]A triangle-shaped formation; more information can be obtained from http://en.wikipedia.org/wiki/Flying_wedge.

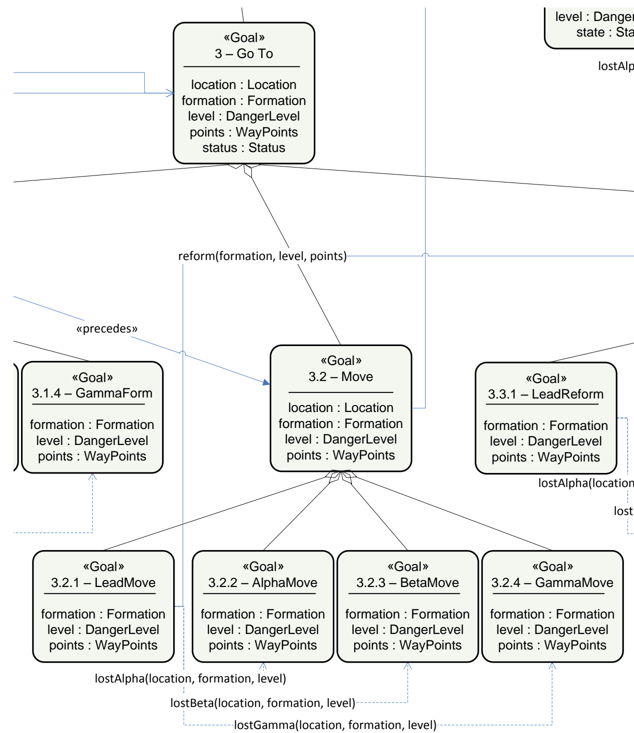


Figure 5.26: Partial Goal Model

changes the danger level from medium to low. The following five figures (Figure 5.27, Figure 5.28a, Figure 5.28b, Figure 5.28c, and Figure 5.28d) show the logs of the sequence of events from the IAL and the four robots that is related to changing the value of the *level* parameter.

Figure 5.27 shows a reformatted and simplified log (from Appendix E) of the IAL. Only the relevant lines from the log that are related to the change of value in the *level* parameter of the *Go To* goal are shown. At line 37, the IAL receives the goal modification from the human supervisor for the *Go To* goal to change the value of *level* parameter. Line 38 lists the four leaf-goals that are affected by this change: *LeadMove*, *AlphaMove*, *BetaMove*, and *GammaMove*. Lines 39–42 lists the four robots that are currently assigned to the respective leaf-goals: *red* robot that is assigned to the *LeadMove* goal, *blue* robot that is assigned to the *AlphaMove* goal, *green* robot that is assigned to the *BetaMove* goal, and *black* robot that is assigned to the *GammaMove* goal.

```
37 2011/05/27 14:18:24 >> Goal Modification (Go To[1])
   status = null,
   points = [(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)],
   Task ID = 1,
   location = recon.data.ReconArea [x=31, y=75, width=0, height=0],
   level = LOW,
   formation = WEDGE

38 2011/05/27 14:18:24 >> Affected Goals
   LeadMove[1]
    state = ([2, 1, 1],false),
    points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)],
    location = *recon.data.ReconArea [x=31, y=75, width=0, height=0],
    level = *LOW,
    formation = *WEDGE
   AlphaMove[1]
    state = ([2, 1, 1],false),
    points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)],
    location = *recon.data.ReconArea [x=31, y=75, width=0, height=0],
    level = *LOW,
    formation = *WEDGE
   BetaMove[1]
    state = ([2, 1, 1],false),
    points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)],
    location = *recon.data.ReconArea [x=31, y=75, width=0, height=0],
    level = *LOW,
    formation = *WEDGE
   GammaMove[1]
    state = ([2, 1, 1],false),
    points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)],
    location = *recon.data.ReconArea [x=31, y=75, width=0, height=0],
    level = *LOW,
    formation = *WEDGE

39 2011/05/27 14:18:24 >> Goal (BetaMove[1]) Assigned To Agent (green)

40 2011/05/27 14:18:24 >> Goal (LeadMove[1]) Assigned To Agent (red)

41 2011/05/27 14:18:24 >> Goal (AlphaMove[1]) Assigned To Agent (blue)

42 2011/05/27 14:18:24 >> Goal (GammaMove[1]) Assigned To Agent (black)
```

Figure 5.27: Reformatted Log of IAL

The IAL informs the four robots about the change in their respective goals. The following figure (Figure 5.28) shows how each robot receives and reacts to the change in value for the *level* parameter. Figure 5.28a shows the *red* robot receiving the change for the *level* parameter. However, because the *red* robot is playing the leader role in the formation, it does not need to react to this change. Figure 5.28b shows the *blue* robot receiving the change for the *level* parameter. When the *blue* robot, which is playing the alpha position in the formation, receives the change, it reacts by changing the distance that it needs to be from the leader position from 1100 millimeters to 1000 millimeters. Figure 5.28c shows the *green* robot receiving the change for the *level* parameter. When the *green* robot, which is playing the beta position in the formation, receives the change, it reacts by changing the distance that it needs to be from the leader position from 1100 millimeters to 1000 millimeters. Figure 5.28d shows the *black* robot receiving the change for the *level* parameter. When the *black* robot, which is playing the gamma position in the formation, receives the change, it reacts by changing the distance that it needs to be from the alpha position from 1100 millimeters to 1000 millimeters.

The next goal modification that the human supervisor changes is the waypoints. The human supervisor decides to add a fourth waypoint to the original three waypoints. Figure 5.29a shows the interface before the modification of the *points* parameter of the *Go To* goal and Figure 5.29b shows the interface after the modification. The following five figures (Figure 5.30, Figure 5.31a, Figure 5.31b, Figure 5.31c, and Figure 5.31d) show the logs of the sequence of events from the IAL and the four robots that is related to changing the value of the *points* parameter.

Figure 5.30 shows a reformatted and simplified log (from Appendix E) of the IAL. Only the relevant lines from the log that are related to the change of value in the *points* parameter of the *Go To* goal are shown. At line 43, the IAL receives the goal modification from the human supervisor for the *Go To* goal to change the value of the *points* parameter. Line 44

```
5   2011/05/27 14:18:25 >> Goal Modification (LeadMove[1])
     formation = *WEDGE,
     level = *LOW,
     state = ([2, 1, 1],false),
     location = *recon.data.ReconArea [x=31, y=75, width=0, height=0],
     points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)]
```

(a) “red” Robot

```
4   2011/05/27 14:18:24 >> Goal Modification (AlphaMove[1])
     state = ([2, 1, 1],false),
     level = *LOW,
     location = *recon.data.ReconArea [x=31, y=75, width=0, height=0],
     points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)],
     formation = *WEDGE

5   2011/05/27 14:18:24 >> Reaction To Modification
     Previous Value: (MEDIUM), Distance Apart: (1100)
     New Value: (LOW), Distance Apart: (1000)
```

(b) “blue” Robot

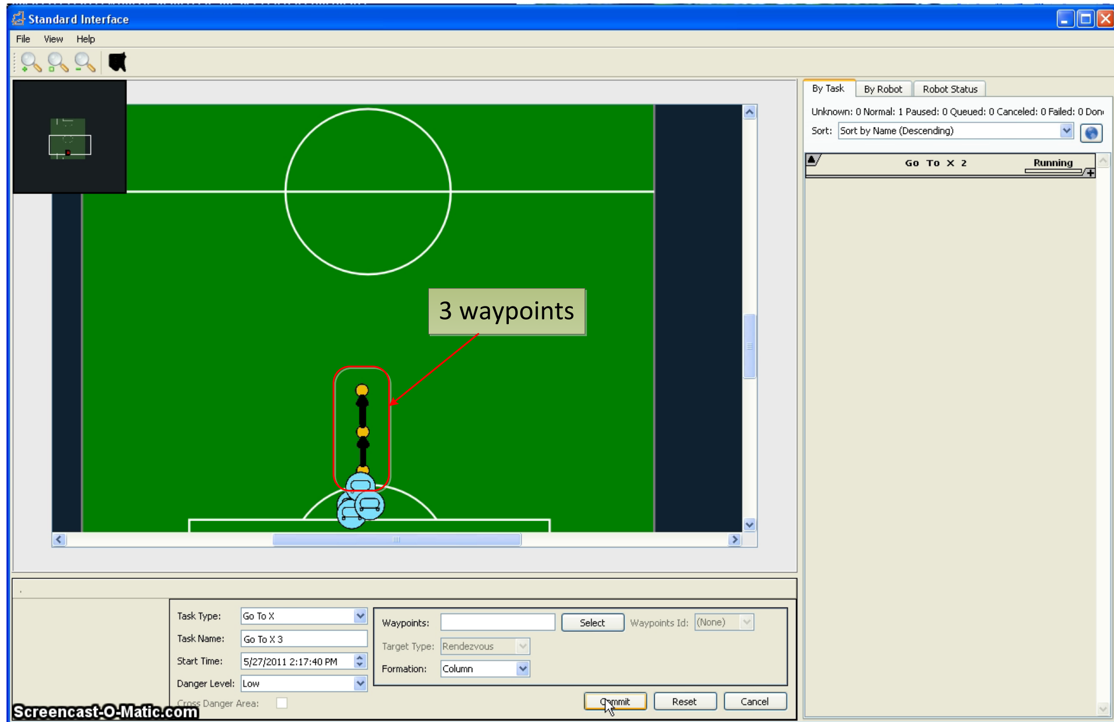
```
4   2011/05/27 14:18:25 >> Goal Modification (BetaMove[1])
     location = *recon.data.ReconArea [x=31, y=75, width=0, height=0],
     state = ([2, 1, 1],false),
     level = *LOW,
     points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)],
     formation = *WEDGE

5   2011/05/27 14:18:25 >> Reaction To Modification
     Previous Value: (MEDIUM), Distance Apart: (1100)
     New Value: (LOW), Distance Apart: (1000)
```

(c) “green” Robot

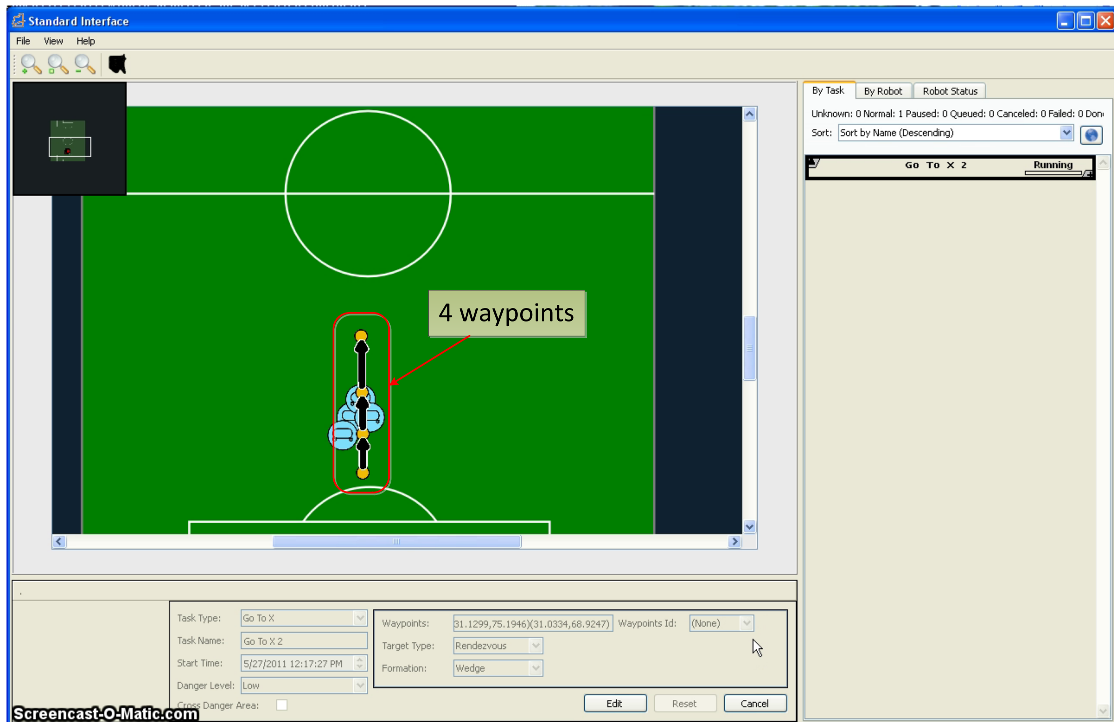
```
4   2011/05/27 14:18:28 >> Goal Modification (GammaMove[1])
     points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)],
     location = *recon.data.ReconArea [x=31, y=75, width=0, height=0],
     state = ([2, 1, 1],false),
     level = *LOW,
     formation = *WEDGE

5   2011/05/27 14:18:28 >> Reaction To Modification
     Previous Value: (MEDIUM), Distance Apart: (1100)
     New Value: (LOW), Distance Apart: (1000)
```

(d) “black” Robot

Figure 5.28: Reformatted Log of Robots



(a) Before



(b) After

Figure 5.29: Waypoint Modification

lists the four leaf-goals that are affected by this change: *LeadMove*, *AlphaMove*, *BetaMove*, and *GammaMove*. Lines 45–48 lists the four robots that are currently assigned to the respective leaf-goals: *red* robot that is assigned to the *LeadMove* goal, *blue* robot that is assigned to the *AlphaMove* goal, *green* robot that is assigned to the *BetaMove* goal, and *black* robot that is assigned to the *GammaMove* goal.

The IAL informs the four robots about the change in their respective goals. The following figure (Figure 5.31) shows how each robot receives and reacts to the change in value for the *points* parameter. Figure 5.31a shows the *red* robot receiving and reacting to the change for the *points* parameter. Line 6 is when the *red* robot receives the change. Lines 3, 7, and 9 show the *red* robot indicating that the team has reached a waypoint; line 9 is the new fourth waypoint added by the human supervisor. Figure 5.31b shows the *blue* robot receiving the change for the *points* parameter. However, because the *blue* robot is simply following the *red* robot, it does not need to react to this change. Figure 5.31c shows the *green* robot receiving change for the *points* parameter. However, because the *green* robot is simply following the *red* robot, it does not need to react to this change. Figure 5.31d shows the *black* robot receiving the change for the *points* parameter. However, because the *black* robot is simply following the *blue* robot, it does not need to react to this change.

The proof-of-concept demonstration shows the usefulness of enabling a human supervisor to modify goals, especially for team goals. The next section (Section 5.4) summarizes this chapter.

5.4 Summary

In summary, this chapter defines organization control and a set of interactions for organization control. In addition, an architecture is presented that shows how the IAL (Section 5.2) can autonomously perform task allocation and adapt to failures. The IAL frees

```
43 2011/05/27 14:18:30 >> Goal Modification (Go To[1])
   status = null,
   points = [(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
   Task ID = 1,
   location = recon.data.ReconArea [x=31, y=69, width=0, height=0],
   level = LOW,
   formation = WEDGE

44 2011/05/27 14:18:30 >> Affected Goals
   BetaMove[1]
              state = ([2, 1, 1],false),
              points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
              location = *recon.data.ReconArea [x=31, y=69, width=0, height=0],
              level = *LOW,
              formation = *WEDGE
   LeadMove[1]
              state = ([2, 1, 1],false),
              points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
              location = *recon.data.ReconArea [x=31, y=69, width=0, height=0],
              level = *LOW,
              formation = *WEDGE
   AlphaMove[1]
              state = ([2, 1, 1],false),
              points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
              location = *recon.data.ReconArea [x=31, y=69, width=0, height=0],
              level = *LOW,
              formation = *WEDGE
   GammaMove[1]
              state = ([2, 1, 1],false),
              points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
              location = *recon.data.ReconArea [x=31, y=69, width=0, height=0],
              level = *LOW,
              formation = *WEDGE

45 2011/05/27 14:18:30 >> Goal (BetaMove[1]) Assigned To Agent (green)

46 2011/05/27 14:18:30 >> Goal (LeadMove[1]) Assigned To agent (red)

47 2011/05/27 14:18:30 >> Goal (AlphaMove[1]) Assigned To Agent (blue)

48 2011/05/27 14:18:30 >> Goal (GammaMove[1]) Assigned To Agent (black)
```

Figure 5.30: Reformatted Log of IAL

3 2011/05/27 14:18:20 >> Arrived At Waypoint (31.2263, 79.8246, 0.0)

6 2011/05/27 14:18:31 >> Goal Modification (LeadMove[1])
formation = *WEDGE,
level = *LOW,
state = ([2, 1, 1],false),
location = *recon.data.ReconArea [x=31, y=69, width=0, height=0],
points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)]

7 2011/05/27 14:18:33 >> Arrived At Waypoint (31.1299, 75.1946, 0.0)

9 2011/05/27 14:18:53 >> Arrived At Waypoint (31.0334, 68.9247, 0.0)

(a) "red" Robot

6 2011/05/27 14:18:30 >> Goal Modification (AlphaMove[1])
state = ([2, 1, 1],false),
level = *LOW,
location = *recon.data.ReconArea [x=31, y=69, width=0, height=0],
points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)]
formation = *WEDGE

(b) "blue" Robot

6 2011/05/27 14:18:31 >> Goal Modification (BetaMove[1])
location = *recon.data.ReconArea [x=31, y=69, width=0, height=0],
state = ([2, 1, 1],false),
level = *LOW,
points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)]
formation = *WEDGE

(c) "green" Robot

6 2011/05/27 14:18:33 >> Goal Modification (GammaMove[1])
points = *[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
location = *recon.data.ReconArea [x=31, y=69, width=0, height=0],
state = ([2, 1, 1],false),
level = *LOW,
formation = *WEDGE

(d) "black" Robot

Figure 5.31: Reformatted Log of Leader Robot

up a human supervisor from the need to micromanage the assignments. The IAL also allows a human supervisor to exercise organization control. The two interactions for manipulating assignments and a demonstration of human supervisor assignment manipulation is presented in Section 5.3.1. Section 5.3.2 describes a formal extension to GMoDS to allow goal modification, proves the correctness of the formalization, and demonstrates its usefulness.

Role-related interactions for organization control are not addressed because there is currently no autonomous means of generating role behaviors other than writing executable code and not all human supervisors are proficient at writing executable code. Furthermore, code injection for new roles or a means for translating role behavior to executable code is beyond the scope of this dissertation.

In addition, the goal creation and removal interactions are not addressed because these would require extensions to GMoDS similar to goal modification. But unlike goal modification, which adds new formalizations to GMoDS and does not modify any existing formalization in GMoDS, the formalization for goal creation and removal would require changes to existing formalization in GMoDS.

The next chapter (Chapter 6) discusses related work and how they compare to the work in this dissertation.

Chapter 6

Related Work

This chapter looks at existing work that are related to my work. Section 6.1 looks at work related to multirobot task allocation. Section 6.2 looks at related work in applying PMFs. Section 6.3 looks at work that are aimed at increasing the human-to-robot ratio of supervisory control.

6.1 Task Allocation

In multirobot systems, Parker [72] defined three approaches to tackling the problem of task allocation: bioinspired, organizational, and knowledge-based. Furthermore, Gerkey and Matarić [42] provided a taxonomy for multirobot task allocation problems. The four classifications are: (1) *single-task robot*, where a robot can perform at most one task at a time; (2) *multi-task robot*, where a robot can perform multiple tasks simultaneously; (3) *single-robot task*, where a task requires exactly one robot; (4) *multi-robot task*, where a task requires multiple robots.

6.1.1 Bioinspired Approaches

In bioinspired approaches, observations made on animal/insect behaviors are applied to solve the problem of task allocation in multirobot systems. A commonly used behavior is from the study of ants; the most popular application of ant behavior is the Ant Colony Optimization (ACO) [34] technique, which was inspired by the foraging behavior of ants. Similarly, some animal/insect behaviours can be applied to the task allocation problem in multirobot systems. Robots in bioinspired approaches are typically homogeneous and exists in large numbers (i.e., swarms). Individually, each robot possesses very limited capabilities. However, when they are grouped together in swarms and interact as a collective, a group-level intelligent behavior emerges. Because it is assumed that every robot has the ability to sense the relevant information in their environment (i.e., *stigmergy* [58]), communication among the robots is reduced significantly. Even in situations when *stigmergy* is not available, robots only need to broadcast minimal information about their state or environment. Because of the minimal communication, there is no need for the robots to communicate about task allocation. A task is allocated when a robot senses that a task needs to be performed and proceeds to perform it. Should a robot fail when performing a task, another robot simply replaces the failed robot. By following this basic behavior, a collective of these robots can achieve the overall system goal.

The following works are broadly equivalent to the above description on bioinspired approaches. The differences lie in the details such as the architecture, framework, and/or type of robot deployed. For example, Stilwell and Bay [91] proposed a decentralized architecture for controlling a swarm of homogeneous robots in transporting materials using a pallet. A robot is much smaller in size than a pallet and the maximum carrying weight of a robot is significantly lower than the weight of the pallet. However, as a collective, the weight can be distributed among the robots so that no single robot is over the maximum carrying weight. Furthermore, the robots know the direction in which to move the pallet. Because

the robots are homogeneous and able to determine their placement through *stigmergy*, no direct communication is necessary to coordinate the robots.

Matarić [58] proposed an approach for using a group of homogeneous robots in a foraging scenario. A robot can exhibit one of the six basic behavior (collision avoidance, following, dispersion, aggregation, homing, and flocking) when looking for food or carrying the food home. Again, no direct communication is necessary because the robots are homogeneous and use *stigmergy*.

Kube and Zhang [52] proposed an approach for using a group of homogeneous robot in a box-pushing scenario. Each robot can exhibit one of the five behaviors: (1) move to destination, (2) avoid collisions, (3) follow another robot, (4) slow down to avoid rear-end collisions, and (5) find. The authors experimented with two different strategies in behavior selection: fixed priority behavior preference and a neural network. Again, no direct communication is necessary because the robots are homogeneous and use *stigmergy*.

Balch and Arkin [6] proposed a schema-based reactive control system for controlling a group of homogeneous robots. A schema describes a task in terms of states and associated behaviors. The authors defined three schemas for three tasks: forage, consume, and graze. Their system is able to function without direct communication for the three tasks but the authors have shown that state communication significantly improves the performance of the forage and consume tasks but is unnecessary for the graze task. The reason for the performance gain is because forage and consume tasks have little impact on the environment. The graze task on the other hand have an impact on the environment, which can be sensed by other robots.

Kubo and Kakazu [53] proposed a reactive planning system for controlling a group of homogeneous robots in a competitive scenario of foraging for food. The authors applied reinforcement learning (particularly, the stochastic learning automata) to learn strategies to deal with the opposing team.

Kazuo and Suzuki [93] proposed a distributed algorithm for controlling a group of homogeneous robots in forming geometric shapes such as circles and polygons. The algorithm assumes that all robots has the same physical characteristics and abilities. Again, no direct communication is necessary because the robots are homogeneous and *stigmergy*.

Sun, Lee, and Sim [94] proposed an approached based on the human immune system for controlling a group of homogeneous robots. The authors applied their approach to four tasks: aggregation, random search, dispersion, and homing. There is some communication involved such as state information being exchanged. However, these communications are localized to robots that are in close proximity (i.e., local inter-robot communication).

Passino [73] proposed an algorithm based on the foraging behavior of bacteria and showed how the algorithm can be viewed as an optimization algorithm to provide adaptive control on problems that can be transformed to an optimization problem such as autonomous guidance of AVs.

McLurkin and Smith [60] proposed a directed dispersion algorithm for controlling a group of homogeneous robots in exploring large and complex indoor environments. The directed dispersion algorithm was designed to spread the robots out quickly and uniformly in an enclosed space while also maintaining a communication network. The communication network is formed through a series of local inter-robot communication.

A major criticism of bioinspired approaches is the dependence on homogeneous robots. Because of the assumption that the robots are homogeneous, there is no need to provide any mechanisms for determining what a robot is capable of performing. This dependence typically results in a solution to a particular problem. While it is possible that a given solution can be re-engineered to solve another problem, the re-engineering process typically include reworking the core aspects of the existing solution such as using a different type of robot and figuring out new algorithms.

Another criticism of bioinspired approaches is that solutions addresses a particular

problem and the scope of the problem is typically limited. One of the reasons is because of cost. As the scope of a problem increases, the number of different types of tasks also increases. Since the robots are homogeneous, this means that the robots must be able to perform all those tasks. Building a homogeneous robot that can perform all tasks is more expensive than building different robots that can perform a subset of those tasks.

6.1.2 Organizational Approaches

Organizational approaches utilize organizational theory for task allocation in multirobot systems. There are two sub-approaches in organizational approaches: role-based and market-based.

6.1.2.1 Role-Based Approaches

Role-based approaches employ the use of roles to divide up the work that needs to be done. A role can consists of one or more tasks that need to be completed. Once the set of roles have been defined, robots select (or are assigned) the roles that are best suited for them. Pure role-based approaches typically predefine the set of agents that can perform a particular role; thus there is no need to determine at runtime the set of agents that can perform a particular role. Role-based approaches that that determine role-agent mappings at runtime typically employ ontology/semantic information. For the purposes of clarity, any role-based approaches that use ontology/semantic information in the task allocation process are listed in the next section (6.1.3). The following work are examples of pure role-based approaches.

Stone and Veloso [92] proposed an architecture for use in periodic team synchronization domains. A periodic team synchronization domain is defined as consisting of a team of autonomous agents operating in an environment where communication is unreliable but periodically communication will be unrestricted and unlimited. One example of a periodic team synchronization domain is robotic soccer. In the architecture, roles specify the internal

and external behavior of agents such as the captain's position on the soccer team. The robot in the authors' experiment are homogeneous and capable of performing any of the defined roles for the robotic soccer domain. An agent starts out with the initial role it is playing and a set of plays (which are strategies that defined the actions/behaviours that should be carried out at particular situations). Once the game starts, communication is unreliable and limited. Agents will have to infer what roles their teammates are playing so that they can infer the current strategy that the team is executing, which determines the current roles that agent should be playing. A major limitation of the authors' architecture is the assumption that all agents are capable of performing any role, which inherently implies that the agents are homogeneous.

Simmons *et al.* [89] proposed a role-based approach and applied it to the assembly (i.e., building large structures) domain. The assembly domain consists of three robots: (1) a crane that is able to carry and move heavy beams, (2) a roving eye that uses stereo cameras to provide a fine-grained positions of beams, and (3) a mobile manipulator that uses an arm to perform fine-grained adjustments of beams. The three robots and the "foreman" agent will perform the task of assembling a large structure. In the authors' domain, each of the robot also represent a role: (1) crane, (2) roving eye, and (3) mobile manipulator. In the authors' approach, there is no need to determine at runtime which robot can perform which role since a robot is equivalent to a role. Although the assembly domain can be fairly complex, the setup is relatively simple as it consists of three robots that perform different functions with no overlap. In a more complex setup such as more robots with some overlap, it is uncertain what mechanism is responsible for selecting the group of robots to perform the assembly task.

6.1.2.2 Market-Based Approaches

Market-based approaches use principles and theories of market economies to enable robots to negotiate with other robots on which tasks they should perform. Most approaches use a utility function and/or a cost function for computing the approximate values to performing some action. Once the values for a given task have been computed, robots with the highest (utility) or lowest (cost) value would be the robot performing the task. The following work are broadly equivalent to the above description on market-based approaches. The differences lie in the utility/cost functions, the communication protocols, and the architecture.

Botelho and Alami [9] proposed the M+ architecture. In M+, a task is defined as a set of goals to be achieved. For a given task, a robot can decompose the task into a set of actions that the robot can perform to achieve the goals associated with the task. Different robots can decompose the same task into different sets of actions. Each robot has access to the list of tasks that needs to be completed. A robot picks the task with the lowest cost and announces it to the group. If no other robot has selected the task, that robot continues to execute the task. However, if another robot also selects the task, the negotiation process is initiated between the robots to decide the best robot. The M+ architecture shares some similarities to the OBAA architecture. A task is similar to a role in which the role specifies how to achieve a goal. The OBAA architecture also specifies that a role have one or more plans that can be executed. A key difference is that in M+, the set of actions can be autonomously computed at runtime, while plans in OBAA are defined at design time. In M+, tasks are assumed to be achievable by a single robot (i.e., single-robot task). However, a robot can ask other robots for help, which also initiates a negotiation to pick the best robot. This particular feature is not present in OBAA-based systems. However, an equivalent process occurs; if an agent cannot complete a role, an agent failure occurs. The agent failure causes the goal (associated with the role) to be reassigned to another agent.

Gerkey and Matarić [41] proposed the MURDOCH system. MURDOCH is an auction-

based system. In MURDOCH, a task contains metric(s) that robots can use to compute their task-specific fitness. There is an auctioneer agent that is responsible for (1) informing robots of new tasks, (2) collecting bids (in the form of the fitness score) from robots, (3) informing the winner, and (4) monitoring the progress tasks. The winning robot has a time limit in completing the awarded task. The time limit is MURDOCH's way to handle failures. If the auctioneer agent has not seen sufficient progress or has not received any response from the robot performing the task, the auctioneer agent will initiate the auction process. Similar to CzM/OMACS-based systems, MURDOCH performs task allocations for single-robot tasks.

Zlot and Stentz [116] proposed an auction-based system that performs task allocation for complex tasks. A complex task is a task that can be decomposed into smaller subtasks through AND/OR decomposition. The system is an extension to TraderBots [29] to incorporate task decomposition into the allocation process, which enables task allocation from any level of a task tree. Agents/robots bid (cost or utility) on any task(s) that they are able to perform. If the task is a complex task, the winner decomposes the task and may subcontract the subtasks to other agents/robots. If another agent/robot comes along and has a better bid for that complex task, that agent/robot takes over that complex task and may have its own task tree that is different. Incorporating task decomposition into the task allocation process allows a more robust system than the decompose-then-allocate process (which CzM/OMACS-based systems follow). A limitation of the decompose-then-allocate process is that sometimes the decomposed tasks may not be achievable for a group of agents/robots although that group of agents/robots may still achieve the root task if the tree was decomposed differently. There is currently no known algorithm that can perform task decomposition for any task; thus, task decomposition is typically predefined for a known set of tasks. In situations where the tasks are not known a priori, a human expert is required to decompose tasks properly.

Vig and Adams [102] proposed the RACHNA architecture. In RACHNA, there are two

types of agents[1]: service agents and task agents. A service agent represents a role or service that a robot can perform or provide (i.e., if there are six services that a robot can provide, then there are at least six service agents). A task agent represents a task (which represents a list of required services) that needs to be completed. A task agent bids on services that the task requires (by communicating with service agents). Since a service agent maintains a list of robots that can provide a particular service, the service agent will respond to bids with the robot(s) that accepted the bids. Once an auction is successfully completed, the task agent can communicate directly with robots that are going to be providing the necessary services to complete the task. Tasks in RACHNA are multi-robot tasks, which CzM/OMACS-based systems cannot handle.

Dash *et al.* [21] proposed an auction-based system for self-interested agents. A self-interested agent may lie about its capabilities so that it may avoid undesirable tasks. There are two implementations of the system: a centralized system and a billboard-type decentralized system. The centralized system includes a penalty mechanism that provides an incentive for agents to truthfully report their capabilities. In other words, the penalty mechanism ensures that it is in the best interest of the agents to truthfully reveal private information. The billboard-type system is a complimentary approach to the centralized approach but does not always result in the best allocations. OBAA-based systems typically assume cooperative and honest agents and little work has been done in incorporating self-interested agents. Although it may be possible to provide an IAL that allows self-interested agents, it remains to be seen if the existing architecture is sufficient.

Vincent *et al.* [103] proposed an algorithm for searching for objects of interest and protecting objects of interest. The algorithm can work in one of two modes: managed, or auction-based. In the managed mode, a dispatcher maintains the necessary information about each robot such as position and battery level. Based on that information, the

[1]In the author's context, an agent is purely software based and is not synonymous to a robot.

dispatcher informs the robots of their assigned task. The process of the managed mode is similar to the centralized implementation of the IAL, where one of the CC is the master and the rest are slaves. In the auction-based mode, robots receive a list of tasks, ranks the tasks, and return the ranking to the dispatcher. Once the dispatcher has the rankings, the dispatcher allocates tasks based on the rankings. Although my work uses a centralized implementation of the IAL, because of the flexibility of the OBAA architecture, a different implementation such as an auction-based IAL can be substituted for the centralized implementation without interfering with the rest of the system.

Choi, Brunet, and How [14] proposed two algorithms: consensus-based auction algorithm and consensus-based bundle algorithm. The consensus-based auction algorithm is for single-task agents/robots, while the consensus-based bundle algorithm is for multi-task agents/robots. In situations where it is not feasible to ensure synchronized state information across all agents/robots, an agent/robot may have information about the environment that is inconsistent with another agent/robot. This inconsistencies in information may result in different assignments. By combining an auction-based approach for assignments with a consensus-based approach for conflict resolution of assignments, the two algorithms are able to function in environments where the information of agents/robots about the environment could be inconsistent. Although conflict resolution is not the focus of my research, due to the flexibility of the OBAA architecture, it may be possible to implement an IAL (particularly the ORC) that can function in environments where agents may not have the same information about the environment as other agents.

6.1.3 Knowledge-Based Approaches

Knowledge-based approaches share ontological and/or semantic information among the robots as the basis for task allocation. Typically, these ontological and/or semantic information have some relation to tasks. Through the process of sharing these ontological

and/or semantic information, robots can obtain enough information about other robots to help compute the appropriate robot for a given task. One type of ontological and/or semantic information that is typically shared is the capabilities of robots. Because the capabilities of robots is one of the types of information being shared, knowledge-based approaches are able to include heterogeneous robots when allocating tasks.

The following work are broadly equivalent to the above description on knowledge-based approaches. The difference lie in how these ontological and/or semantic information is captured and how they are applied in task allocations. For example, Parker [71] proposed the ALLIANCE architecture. In ALLIANCE, every robot contains a model of every other robot. The model contains information about the performance of the robots and tasks-related information. Information obtained from observations (typically through the sensors) on the environment is used to populate the models. Then, the robots are able to use their models to determine which task(s) to perform. In ALLIANCE, the models are similar in concept to CzM/OMACS-based system (where every agent contains a model representing other agents) and GMoDS (where events that occur in the environment are processed by GMoDS). Although the ALLIANCE architecture has limited communications, CzM/OMACS-based system depends on the IAL (particularly the ORC) for sharing the necessary information to other agents.

Fua and Ge [37] proposed the COBOS scheme. In COBOS, a team of robots is captured by a task suitability matrix for the purposes of computing task allocation. The task suitability matrix maintains the suitability of each robot for each task. The suitability (i.e., how qualified) of a robot for a given task is based on the capabilities of the robot and environmental factors such as distance of the robot to the task. Each robot contains a task suitability matrix and updates to the matrix are obtained from broadcasts messages, which include a robot's suitability for each task. In COBOS, loss of communication does not imply failure of robot. When a robot loses communication, information pertaining to that robot

(such as task suitability and the task(s) the robot is performing) is left unchanged until communication is reestablished. Furthermore, COBOS is able to handle tasks that have uncertain requirements that result in an unknown number of robots required to complete that task. When the first robot performs a task with uncertain requirements and discovers that more robots are needed, a secondary algorithm is activated to recruit more robots for that task. The task suitability matrix is similar to CzM/OMACS. In CzM/OMACS, capabilities of agents are stored as well as the relationship of those capabilities to roles and goals; based on that information, a score is computed to indicate how well agents can perform roles to achieve goals. Although the issue of communication loss is not specifically addressed in my work, it can be addressed in the ORC, which is responsible for the team coordination aspects of an OBAA-based system such as what happens when an agent joins/leaves the team.

Tang and Parker [95] proposed the ASyMTRe methodology. The basic building block of an ASyMTRe system are environmental sensors and schemas, which functions like component with inputs and outputs. In ASyMTRe, there can be multiple inputs but one output is assumed. There are three types of schemas: perceptual, communication, and motor. The output of environmental sensors can be connected to perceptual schemas. The inputs for perceptual schemas can be from either environmental sensors or communication schemas and the output can go to either communication or motor schemas. The inputs for communication schemas can be from either perceptual or communication schemas and the output can go to either perceptual, communication, or motor schemas. The inputs for the motor schemas can be from either perceptual or communication schemas and the output goes to the robot effector control process. The definition of a task includes a set of motor schemas. ASyMTRe uses the set of motor schemas and attempts to make the necessary connections. When a valid flow is established, that flow represents how and who will attempt to complete the task. The initial reasoning responsible for creating valid

flows in ASyMTRe is centralized but was addressed in later work (ASyMTRe-D) [96]. However, the solution quality in ASyMTRe-D is lower than the centralized version. In another later work, Zhang and Parker proposed the IQ-ASyMTRe [112, 113] architecture, which expands on ASyMTRe, to deal with sensor constraints that are imposed as a result of sharing capabilities. IQ-ASyMTRe addresses the issue of finding a valid solution while satisfying the sensor constraints. ASyMTRe does not restrict task allocations to single-robot tasks but allows multi-robot tasks. In contrast, CzM/OMACS-based system restricts allocations to single-robot tasks; the decomposition of multirobot tasks is done by a designer with GMoDS. Furthermore, an ASyMTRe solution includes the "how" to complete a given task, whereas in CzM/OMACS-based system, the roles are specified at design time.

Vig and Adams [101] provide a heuristic-based coalition (i.e., a group of robots) formation algorithm for independent tasks. Tasks are specified in terms of required capabilities and each robot is represented by a capability vector (which is similar to CzM/OMACS-based systems). The authors introduced the idea of coalition imbalance, which occurs when an agent possesses a significant number of capabilities in comparison to other agents in the same coalition. Coalition imbalance has a negative impact on the fault tolerance of the coalition. The algorithm selects the coalition with the highest fault tolerance, which also favors balanced coalitions. Again, CzM/OMACS-based systems do not allocate multi-robot tasks; multi-robot tasks are decomposed through GMoDS.

Macarthur *et al.* [57] proposed a distributed algorithm (branch-and-bound fast-max-sum) that is based on prior work (fast-max-sum [77]), and fast-max sum is also based on prior work (max-sum [2]). The branch-and-bound fast-max-sum includes two important features. The first feature is an online pruning algorithm that reduces the complexity of the problem and the second feature is the branch-and-bound algorithm that reduces the search space. Together, the two features allow the branch-and-bound fast-max-sum algorithm to scale well when there is a large number of agents and multi-robot tasks. However, the

branch-and-bound fast-max-sum algorithm assumes that an agent can only be assigned to one group task (i.e., single-task robot); whereas, CzM/OMACS-based systems allow agents to be assigned multiple tasks (i.e., multi-task robot). However, CzM/OMACS-based systems only compute assignments for single-robot tasks while the branch-and-bound fast-max-sum algorithm compute assignments for multi-robot tasks.

Vincent *et al.* [103] proposed a task allocation algorithm for exploration and mapping. The algorithm uses a utility function and a cost function to maximize the utility minus the cost. Because the algorithm assigns a robot to one task (i.e., single-task robot), an optimal set of assignments can be computed using a liner program solver fairly quickly ($O(rt)$, where r is the number of robots and t is the number of tasks). The utility function is similar to the `goodness` function in CzM or `rcf` function in OMACS. Although there is no cost function in CzM, the `goodness` function allows consideration of team-based qualities that affect the performance of individual agents. Furthermore, because CzM/OMACS-based systems allow multi-task agents, it is not feasible to find the optimal assignments during runtime ($O(2^{at})$, where a is the number of agents and t is the number of tasks).

Tsalatsanis, Yalcin and Valavanis [98] proposed an approach that is based on supervisory control theory to facilitate task allocation. A robot team is modeled as a Discrete Event System (DES), robots are modeled as a finite state automatons, and task requirements are modeled as discrete events. Because the robots are heterogeneous, a utility function based on sensor capabilities is defined to compute the values of robots for tasks (which is similar to the `goodness` function in CzM or `rcf` function in OMACS). Two types of failures are considered: (1) temporary failures, where a robot loses some capability but can be quickly repaired; and (2) failures, where capability loss cannot be repaired or repairs would take a considerable amount of time. For temporary failures, the robot that failed is considered offline and events pertaining to the tasks that are allocated to that robot are not masked while the robot is being repaired. When repairs are done, the robot resumes the tasks it

has been allocated and the associated events are unmasked. For non-temporary failures, the robot is also considered offline and events are masked. However, the tasks that are allocated to that robot are reallocated to another robot. My work does not specifically address temporary failures but treat failures similar to the latter and reassigns any assignments for the failed agent. However, temporary failures can be implemented as part of the IAL (particularly the ORC).

Dahl, Matarić, and Sukhatme [19] proposed a distributed algorithm (called vacancy chain scheduling) for a prioritized transportation scenario. An example of a vacancy chain is how a family car is passed on; the parents buy a new car to replace an existing one, the existing car is passed to the oldest child, and if that child has an existing car, that car is also passed on, and so forth. The vacancy chain scheduling algorithm has no explicit communication but instead relies on stigmergy. For the algorithm to work, tasks must be spatially classifiable, so that robots can compute the local utility estimates. In addition, vacancy chain scheduling is inherently limited to single-robot task (similar to CzM/OMACS-based system). Furthermore, vacancy chain scheduling is able to model group dynamics (which are positive or negative effects on performance when members of a group interact). Although CzM is able to model some group dynamics indirectly through the `goodness` function, it remains to be seen if the `goodness` function is sufficient for modeling all group dynamics.

6.2 Performance Moderator Functions (PMFs)

PMFs are quantified human performance factors. Some PMFs can be used to model human behavior. Modeling human behavior is vital in realistic military human-based simulations that are used to train future soldiers because good PMFs are accurate approximations of actual human behaviours. A challenge in developing realistic human-

based simulation is validating PMFs [86]. Traditionally, PMFs are validated on a case-by-case basis. However, it is more useful if PMFs can be validated using a unified architecture, which would cut down on the time required to develop simulations to validate PMFs. There are two popular architectures that provide a unified architecture for validating PMFs: Performance Moderator Functions Server (PMFserv) [86] and Improved Performance Research Integration Tool (IMPRINT) [3, 4].

Silverman *et al.* [86] created PMFserv for modeling human behaviours. PMFserv simulates a virtual world that is populated by virtual agents. These virtual agents are given behavioural PMFs that allow the agents to mimic human behavior. In PMFserv, behaviours are broken up into four areas: (1) psychobiological, (2) personality, culture, and affect, (3) social, and (4) cognitive. The psychobiological area deals with motivators and stressors. The personality, culture, and affect area deals with likes/dislikes, preferences, and emotion. The social area deals with the relationships between agents and the cognitive area deals with decision making. The four areas allow a more realistic simulation of human behavior.

IMPRINT, on the other hand, does not simulate a virtual world with virtual agents to validate PMFs. Instead, IMPRINT is a discrete event-based network system. IMPRINT is used to identify system requirements such as personnel and manpower required to effectively maintain and operate a system under various environmental conditions. In IMPRINT, a task network is created, where each task specifies a set of requirements such as the length of time to complete the task, the circumstances that must occur before, during, and after the task, and the personnel that will be performing the task. Furthermore, IMPRINT allows PMFs to be included in one of the four micro-models: (1) perceptual, (2) cognitive, (3) motor, and (4) special. By default, IMPRINT already includes some PMFs that are a constant value in each of the four micro-models. For example, the perceptual micro-model has eye and head movement time; the cognitive micro-model has perceptual, decision, and

motor process time; the motor micro-model has cursor movement using mouse; and the special micro-model has priorities.

PMFserv and IMPRINT are tools that are used to validate PMFs. Although validating PMFs is not the focus of my work, they are complimentary to validating PMFs. Once a PMF is validated, the validated PMF can be capture and used in CzM to reason about the ability of agents to perform roles to achieve goals. However, PMFserv is not just a tool for validating PMFs. Because PMFserv is able to simulate virtual agents, PMFserv is also used to emulate realistic behavior of virtual agents in simulated environments. In 2003, PMFserv was integrated with the original Unreal Tournament[2] to recreate the Black Hawk Down scenario [87]. In subsequent years, PMFserv has been integrated with a number of different simulations.

Pelechano *et al.* [74] proposed an architecture that integrated Multi-Agent Communication for Evacuation Simulation (MACES) with PMFserv. MACES is a crowd simulation system that performs high-level path-finding in exploring unknown environments such as an unfamiliar building in order to find exits during emergencies. Typically, most crowd control simulations have a large number of individuals that have the same behavior; some simulations may offer limited variability in behaviours based on age and gender. However, by integrating with PMFserv, MACES is able to offer a large variety of validated behaviours based on demographics and culture, which can enhance emergent crowd behavior.

Silverman *et al.* [75, 88] proposed an architecture called NonKin Village, where “the player(s) interacts with other virtual or real followers and leaders of contending factions at a local village level”. The purpose of NonKin Village is to provide an immersive training environment through the simulation of insurgent operations in a village. A player can attempt to influence the virtual world through four categories of actions (Diplomatic, Intelligence, Military, and Economic) while being constrained by limited resources. PMFserv

[2]Unreal Tournament 3 (http://www.unrealtournament.com/) is the latest version, which is powered by the Unreal engine (http://www.unrealengine.com/).

is a core part of the architecture that is used for emulating realistic human behavior of the virtual people in the village.

One way of using PMFs is what Silverman is doing with PMFserv. By using validated PMFs, PMFserv is able to emulate realistic human behavior in a virtual environment. My work, on the other hand, uses PMFs in a different way. My runtime model (i.e., the CzM model) captures PMFs and the necessary relationships to tasks and agents. Then these PMFs can be used by task allocation algorithms that may help in making better assignments.

6.3 Supervisory Control

In the area of supervisory control, there is ongoing work that is aimed at addressing the human-to-robot ratio. A user study by Crandall and Cummings [17] suggests that the highest performance is somewhere between four and six robots. Most work are aimed at increasing the number of robots a human supervisor is able to control effectively.

A popular approach is the playbook style for controlling multiple robots. Miller, Pelican, and Goldman [65] introduced the "tasking" interface for controlling Uninhabited Combat Aerial Vehicles (UCAVs). The interface is modeled such that it is similar to how playbooks work in sports. For example, how a coach supervises a team of players or how the quarterback gives orders to other players. There are three requirements [63, 65] that is necessary for "tasking" interfaces to be useful: (1) a shared vocabulary of tasks where humans can issue tasks and a mechanism to know how the tasks are going to be accomplished, (2) enough knowledge from the interface so that intelligent decisions can be made about how to accomplish tasks, and (3) a way to inspect and manipulate the shared vocabulary and an easy and fast way to view the details of the tasks. The Playbook GUI allows a human to plan three types of missions (interdiction, airfield denial, and suppress

enemy air defenses). After selecting the mission type, the human can provide further details (such as route, roles for the UCAVs, etc.) or let the Mission Analysis Component (MAC) complete the details automatically. A goal of "tasking" interfaces is to provide the flexibility for a human to choose a comfortable level of control between the two extremes of robot automation (level 1 versus level 10). In a later work [69], the "tasking" interface was simplified (called delegation-type interface) and tested with RoboFlag [20]. This delegation-type interface retained the ability to give tasks to robots. However, the ability to provide further details were removed from the interface. In the delegation-type interface, the human selects a robot and then selects one of the three defined task (circle offense, circle defense, and patrol border). There is also the option of selecting all robots and then selecting a task. Miller and Parasuraman [64] noted that there have multiple demonstrations of the Playbook style but none of the demonstrations have completely realized the Playbook vision.

The research in Playbook is orthogonal to the research in this dissertation. For instance, the delegation-type interface [69], where the human supervisor selects a robot and then a task, is similar to the concept of assignments in OMACS/CzM. The difference is that, in OMACS/CzM, there is a mechanism (IAL) that autonomously computes the assignments but the human supervisor still retains the ability to modify the assignments should the need arise. In contrast, the "tasking" interface that works in conjunction with the MAC provides the flexibility to "drill down" (i.e., to allow a human supervisor to provide as much or little details for tasks). In OMACS/CzM, the flexibility to "drill down" can be met by a combination of goal parameters and the roles. There is already a mechanism to modify goal parameters. Similarly, roles would need a mechanism to modify the behaviours. However, such a mechanism to modify role behavior is beyond the scope of this dissertation.

Another different approach of research that is attempting to increase the human-to-robot ratio of supervisory control is RoboLeader [13, 90]. RoboLeader is an intelligent agent that was developed to assist humans in route planning. In an early experiment [13], it was

shown that RoboLeader does not improve situation awareness or performance (in terms of target detection). Human operators with higher spatial ability outperformed operators with lower ability regardless if RoboLeader was used. Furthermore, RoboLeader does not improve performance when the number of robots increased (e.g., 4-robots versus 8-robots). However, RoboLeader was able to increase efficiency and thus reduce the completion time of missions. Currently, RoboLeader (as of 2010) can assist human operators in route planning. However, RoboLeader is designed so that additional capabilities can be added. Four capabilities [90] are being explored: (1) asset allocation and mission planning, (2) real-time motion and obstacle avoidance, (3) cooperative control, and (4) decision making models.

Wang *et al.* [105, 106] proposed an asynchronous interface (Image Queue Interface) for an urban search and rescue scenario. In the scenario, a human operator is tasked to identify victims using videos from 12 robots. The asynchronous interface mines the video streams from all the robots and only shows the relevant parts to the human operator. The authors compared their approach to the traditional approach where the human operator watches the 12 video streams and looks for victims. Their results showed that there is no significant difference in terms of performance (i.e., number of victims found). However, there are significant differences (in favor of their approach) in terms of errors (i.e., false positives and negatives) and operator workload. Although the Image Queue Interface serves a specific function, the concept of asynchronous display and aggregating video streams (from multiple sources) into one video stream may exhibit similar be applicable to other types of data such as the set of assignments. For instance, is there just a single list that displays all the assignments or are there multiple lists (one per agent) that shows the assignments of each agent. However, the user interface design aspect is not the focus of my work but the runtime models (both OMACS and CzM) do provide the means to do so.

Scerri *et al.* [80] proposed a framework (Machinetta) that facilitates coordination of a large group of robots (the authors tested up to 200 Uninhabited Aerial Vehicles (UAVs)).

Their framework is separated into two parts: the low-level dynamic control of an UAV and the high-level coordination. The high-level coordination is contained in proxies. Each UAV has a proxy. There are one or more Team Oriented Plans (TOPs) per proxy. A TOP has a similar structure to typical a goal tree, where a leaf node (called role) is assigned to one team member. To ensure coordination, a proxy has a number of coordination agents (which are created when necessary) that fall into three categories: PlanAgents, RoleAgents, and InformationAgents. A PlanAgent is responsible for (1) instantiating and terminating TOPs, (2) creating the associated RoleAgents, (3) resolving conflicting TOPs with other PlanAgents, and (4) ensuring that there is at most one TOP attempting to achieve a goal with other PlanAgents. A RoleAgent is responsible for finding a team member to perform a specific role. An InformationAgent is responsible for sharing information with other proxies, which are created when there is any new information that should be shared. The Machinetta framework is similar to the OBAA architecture. A Machinetta proxy is similar to the CC, where it is responsible for coordinating a team. However, a primary difference is that a Machinetta proxy can have multiple TOPs (one TOP represents a team), which allows multiple teams to be represented in the Machinetta framework. Whereas, the OBAA architecture only has one goal model and one organization model, and thus no proper representation for multiple teams. However, it is uncertain how a supervisor can exert control over a Machinetta system as it has not been specifically addressed by the authors.

McLurkin *et al.* [61] proposed an approach for developing, debugging, and evaluating distributed algorithm on a large swarm of robots (the authors' experiment consists of 112 iRobot SwarmBots). One of the key aspects of their approach is the centralized user interfaces: a terminal display and a GUI display. The terminal display is mostly used for user input. The GUI display shows "real-time telemetry data, detailed internal state, local neighbor positioning, and global robot positioning", which were inspired by real-time strategy games. Another key aspect of their approach is the use of Swarmish language,

which consists of visual and audio outputs. The visual output consists of three Light-Emitting Diodes (LEDs) (red, green, and blue) that are mounted on each robot, which is also shown in the GUI. The LEDs have been programmed to display certain patterns that indicate the current state of a robot, which can be understood by an experienced user in about $\frac{1}{2}$ second. The audio output consists of a simple MIDI player that is also mounted on each robot. The player has been programmed to play a single note that can vary in terms of instrument, pitch, duration, and volume. Once a user has learned the tempo and rhythm of a good run of the overall system, the user should be able to pick up abnormalities, which are indicative of bugs, without the need to perform a detailed analysis of the execution traces of all robots. The focus of their work is on the outputs so that human users are not overwhelmed by the tremendous amount of data coming from all the robots. Furthermore, since their approach is primarily for development, debugging, and evaluation, developers benefit the most as their approach helps developers to quickly spot bugs and deploy fixes easily. In contrast, my research is on a human being able to supervise an OBAA-based system. Also, the robots are the same (i.e., iRobot SwarmBot).

Kira and Potter [50] proposed an approach that consists of two forms of real-time control that a human operator can exert over a swarm of robots: top-down control and bottom-up control. Top-down control allows a human operator to control the overall swarm behavior through the use of swarm characteristics. A swarm characteristic is an abstraction of some lower-level parameters of individual robots. By changing these swarm characteristics, a human operator can control the behavior of the swarm. The bottom-up control allows a human operator to influence a subset of the swarm through the introduction of virtual elements. The swarm would react to these virtual elements as they would if the swarm encounters these elements in the physical world. Through the bottom-up control, a human operator can indirectly control swarms to exhibit desired behaviors. Although their approach can be generalized to other swarm-based system, their approach does not cater to

heterogeneous robots. For example, the top-down approach works because all robots have the same lower-level parameters that can be abstracted to a characteristic and modifying these characteristics controls overall swarm behavior. However, in heterogeneous domains, it is not necessarily the case that all robots have some common lower-level parameters. This limitation results in characteristics that controls a subset of the robots. Similarly, the bottom-up control have the same limitation in the domain of heterogeneous robots. Robots may react differently to the same virtual element that could lead to undesirable outcomes.

Ding *et al.* [32, 33] proposed a framework that provides decision-support to a human operator for controlling multiple UAVs. The framework functions in one of two modes: autonomous mode and pilot-control mode. In both modes, all UAVs are controlled via the leader-follower routine. All UAVs can be designated as either a leader or a follower but there is only one leader at any given time. In the autonomous mode, the human operator can change which UAV is the leader, the system automatically adjusts the formation to the new leader. To obtain more control, the human operator can switch over to pilot-control mode to directly control the leader UAV while the follower UAVs operate autonomously. By controlling the leader, the human operator can exert control over the group. A limitation of their framework is that it does not translate well to heterogeneous domains. While it is possible that there can be multiple leaders for different types of robots, a leader has to be grouped with similar robots. For example, if a group consists of both aerial and ground robots, controlling that group through the leader-follower approach is hard because the aerial robots moves at a much faster speed than the ground robots.

Podnar *et al.* [76] proposed an architecture that provides multiple levels of human control over a team of robots called Multilevel-Autonomy Robot Telesupervision Architecture (MARTA). The levels of control provided by MARTA range from low-level teleoperation to high-level task planning and monitoring. MARTA is broken into five modules: (1) task planning and monitoring, (2) robot telemonitoring, (3) robot team coordination,

(4) telepresence and teleoperation, and (5) robot controller. The task planning and monitoring module provides the information pertaining to the overall system such as maps representing different aspects of an area, locations of robots, navigation paths, and team status. The robot telemonitoring module archives telemetry and images from each robot and presents the information via a dashboard-type display to the human operator. Furthermore, the archives are also used for automated analysis and monitoring. The robot team coordination module decomposes the the high-level instructions from the task planning and monitoring module into robot-specific instructions. The telepresence and teleoperation module provides the human operator the ability to exert direct control over a robot. The robot controller is responsible for managing the robot-specific instructions that are sent from the robot team coordination module and monitoring the executions of those instructions. MARTA and OBAA share some similarities. The task planning and monitoring module and the robot team coordination module provide similar functionality as the CC from OBAA and the robot controller module provides similar functionality as the EC from OBAA. Some of the functionality of the robot telemonitoring module can be captured using the CzM runtime model as attributes. The functionality of the telepresence and teleoperation module is not present in OBAA.

Mau and Dolan [59] proposed a scheduling algorithm, double Shifted Shortest Processing Time (dSSPT), to reduce the downtime of robots. When supervising a robot team, there are times when a robot requires the attention of a human supervisor to continue (i.e., the robot waits until the human supervisor resolves the situation), which is commonly referred to as downtime. Furthermore, when there are multiple robots, there may be times when multiple robots are waiting for the human supervisor simultaneously. dSSPT prioritizes these requests for the human supervisor so that the downtime of robots is reduced. As a result of reduced downtime, missions can be completed faster. Furthermore, as the human supervisor's time is more efficiently managed, the maximum number of robots that can be

supervised can also be increased. dSSPT is effective for traditional HRI systems where a human operator is required to exert direct control such as teleoperation on the robots to correct situations.

Similar to the previous work (dSSPT), Zheng *et al.* [114] also proposed an algorithm to prioritize which robot the human operator should interact with. In their study, the robots are social robots that answer questions raised by customers. In situations when the robot is unable to respond to a question, the human operator intervenes and instructs the robot (typically through teleoperation) with the appropriate response.

Nevatia *et al.* [67] proposed an approach that allows a human supervisor multiple levels of control over a robot team. At the highest level of control, the human supervisor can designate goals to the robot team. And at the lowest level of control, robots can be teleoperated by the human supervisor. A key feature of their approach is the idea of augmented autonomy. At the highest level of control, if the human supervisor does not provide any goals, the robot team autonomously creates goals instead remaining idle. Furthermore, the human supervisor can also override autonomously created goals. At lower levels, the human supervisor can turn on or off specific autonomous functions such as obstacle avoidance. The idea of augmented autonomy is an important aspect of my work. There is GMoDS that handles the creation of new goals based on events that occur in the environment, there is the RA that computes assignments, and there is OMACS/CzM that enables the ability to change assignments.

6.4 Summary

In summary, this chapter highlights related work in the following areas: task allocation algorithms, PMFs, and increasing the human-to-robot ratio of supervisory control. The next chapter (Chapter 7) concludes the work completed for this dissertation.

Chapter 7

Conclusions

This chapter concludes this dissertation by reiterating the current state and problems (Section 7.1), reviewing the work done and my contributions (Section 7.2), describing the limitations of my work (Section 7.3), and highlighting some areas of future work enabled by my work (Section 7.4).

7.1 Current State & Issues

As computing technology advances, more is being demanded from computing systems. In the field of multiagent systems, agents are expected to be more autonomous and intelligent. Furthermore, as the number of agents in a multiagent system increases, users are also expecting more effective ways to maintain control over the system without becoming overwhelmed by the increasing number of agents.

Large and complex multiagent systems, particularly in the field of robotic agents, are hard to control. Often times, multiple trained human operators are required to control a single robotic agent. Robotic agents can be mass produced, but trained human operators are in limited supply. There is an emerging need for a single human to be able to control

multiple robotic agents simultaneously. Ideally, the mechanism that allows a single human operator to control multiple robotic agents should be scalable, whether the human operator is controlling three robots or hundreds of robots. Traditionally, researchers have attempted to address this issue by introducing more automation, which can free up more time for a human to manage multiple robots. However, this approach has its drawbacks. One of them is that if something goes wrong, the automation needs to be temporarily suspended while the human resolves the issue. If all the robots are the same (i.e., homogeneous), it is less of an issue for the human because the human can be trained to resolve issues for that particular type of robot. However, if the robots are not the same (i.e., heterogeneous), the problem is significantly more severe for the human because the human has to be trained to resolve issues for multiple types of robots. It is harder for a human to be an expert in multiple types of robots, more so as the number of types increases.

Another issue that becomes apparent as multiagent systems become larger is that humans are no longer just users, humans become peers working alongside agents, particularly robotic agents. A multiagent system's adaptability is increased because human peers can perform the work when the agents are unable to due to failures. However, there is a vast amount of information pertaining to humans that can be captured so that multiagent systems can work with their human counterparts. One of the first issues to be resolved is to decide the types of information about humans to capture. Often, human performance factors, which are often indicators of performance with respect to task performance, are the type of information that is first captured. These human performance factors are often captured as PMFs. Second, the information should be captured in such a way as to allow dynamic changes to occur because human performance factors often fluctuate over time.

7.2 Work Done & Contributions

There are three major contributions of this dissertation.

First, the CzM model (Chapter 4) is able to capture information about the state of an agent through the *attribute* entity. This is the first step toward including humans as part of a multiagent system. The associated functions (*contains*, *affects*, and *needs*) provide the necessary structures so that multiagent systems can use this new information. In addition, the CzM model can capture PMFs, which can be used for predicting human performance. The usefulness of the CzM model for task allocation algorithms is demonstrated by two scenarios (Section 4.4 and Section 4.5), which show that the CzM model can improve the results of task allocation algorithms.

Second, organization control (Definition 5.1) provides the ability to exercise supervisory control over a multiagent system as an organization, where agents in the multiagent system have autonomy and intelligence. An agent can be a software agent, a robot, or a human. A group of agents form an organization and a supervisor can exert supervisory control by interacting with the organization as a single entity. Section 5.1 defines and describes a set of interactions that can be used to control an organization.

Third, the OBAA architecture is a general architecture for implementing agents and the IAL (Section 5.2) is formed by correctly implementing the CC component of the OBAA architecture across all agents. The IAL is an autonomous and intelligent mechanism that manages the bulk of the work (such as autonomous task allocation) in managing individual agents. A supervisor interacts with the IAL as a single entity to exercise supervisory control through a set of interactions (Section 5.1). There is a demonstration (Section 5.2.2) that shows that the IAL can perform autonomous task allocations as well as handle agent failures autonomously. There are two demonstrations, assignment set manipulation (Section 5.3.1) and goal modification (Section 5.3.2), that illustrate the usefulness of organization control.

A proof-of-concept demonstration of assignment set manipulation shows how a supervisor can modify assignments at runtime through the OMACS/CzM model. A proof-of-concept demonstration of goal modification, which requires an extension to GMoDS (Section 5.3.2.1), shows how a supervisor can modify the parameters of goals at runtime.

In summary, the following are my three major contributions.

- The definition of a runtime model (i.e., CzM) that captures PMFs so that autonomous mechanisms can reason about humans with respect to their ability to perform tasks and multiple demonstrations that show how using the CzM model can lead to better task allocations.
- The definition of organization control and a set of interactions that can be used to implement organization control.
- The definition of an architecture that facilitates organization control and demonstrations of several organization control interactions. The architecture implements a mechanism that autonomously manages a group of robots and allows a human supervisor to exercise supervisory control over the organization through interactions with the organization instead of the individual robots.

7.3 Limitations

There are three limitations of the work in this dissertation.

First, the IAL can be a single point of failure. As demonstrated in this dissertation, should the main agent fail, the entire system fails. The reason for the single point of failure is that the IAL was implemented as a centralized mechanism. Multiagent systems are inherently distributed systems, so it is desirable for the IAL to be distributed. In theory, the IAL can be distributed. However, to ensure deterministic behavior, the current

implementation of the IAL makes a decision based on the latest information in the runtime models (i.e., GMoDS and CzM). Thus, one agent makes all the decisions while the other agents simply forward information to that one agent for processing. However, if *all* agents make decisions, the IAL will require that information in the runtime models be synchronized across all agents, so that all agents will arrive at the same decisions. However, synchronization is an expensive operation, in terms of communication costs, as any change in the runtime models will need to be propagated to all agents before the next change can occur.

Second, the CzM model does not capture the notion of time. The notion of time is important for some human performance factors such as fatigue. Using workload as a counter example, a task typically has a fixed workload value (x). Thus the workload of an agent performing that task is x. Fatigue, on the other hand, is more complex; there is no fixed fatigue value (y) for the duration of a task. There can be a fixed fatigue value (z) at the completion of a task. However, what is the fatigue value when a task is partially complete? If fatigue increases at a uniform rate, then the duration of the task is sufficient to compute the fatigue value at specific time intervals using $\frac{z}{\text{duration}}$. Unfortunately, that is generally not the case in reality. Human performance factors such as fatigue do not always increase or decrease uniformly. In systems where partially completed tasks can be handed off to other agents without losing progress, it is important to know the fatigue value at intermittent time intervals to determine if the remaining work in a task can be best completed using another agent.

Third, the CzM model does not explicitly capture human performance factors associated with the environment. Human performance factors can be classified into three categories: (1) human-specific, (2) task-specific, and (3) environment-specific. Some examples of environment-specific human performance factors are ambient light, day/night, and raining/sunny. Poor lighting conditions can affect the performance of an agent. Although it

is possible to capture environment-specific human performance factors with the task, it is not an effective solution because environment-specific human performance factors is applicable to most (if not all) tasks. If environment-specific human performance factors are captured with tasks, then these environment-specific human performance factors must be duplicated for each task that needs access to that information. Thus, whenever a environment-specific human performance factor changes, every task that capture that environment-specific human performance factor has to be updated accordingly.

7.4 Future Work

The following are some of the future work based on this dissertation.

A scalable distributed IAL. Multiagent systems are inherently distributed systems, so it would be ideal if the IAL can function in a distributed manner without requiring information in the runtime models to be synchronized across all agents before making a decision. A scalable distributed IAL should be able to function with incomplete or possibly outdated information. When a system continues to function with incomplete or possibly outdated information, there will be situations that result in conflicts. Thus, the IAL must provide a mechanism for conflict resolution. Agents should continue to function correctly when faced with incomplete or possibly outdated information and resolve conflicts when necessary.

Capturing the notion of time in the CzM model. For most purposes of task allocation, knowing the final fatigue value (z) at the end of a task is sufficient. However, there are some situations where finer grain information is advantageous. But PMFs can be used for more than just task allocations; PMFs can be used for continuously tracking human performance factors. A possible use of continuous tracking of PMFs is to allow supervisors to adjust system parameters if certain PMFs are above or below a threshold to guide the system along a different path. Another possible use is to autonomously adapt the agents'

behavior based on PMFs. For example, suppose that a human currently has x workload and assigning the human an extra task would push the workload beyond an acceptable threshold. So instead of the human doing everything in the extra task, another agent could help out the human with the extra task, thus alleviating some of the workload on the human. In order to facilitate continuous tracking of PMFs, the CzM model would have to capture the notion of time.

Capture environment-specific human performance factors in the CzM model. In simulations, environment-specific human performance factors are often ignored because they rarely change for duration of a task. But, environment-specific human performance factors do have an effect on performance. The CzM model will need to capture the concept of the environment as well as provide the necessary associations between the *performance function* entity and the environment so that PMFs can use environment-specific human performance factors in their computations.

Simplify the implementation of HRI/HCI through the use of runtime models. Because runtime models can be an effective means of supervisory control (Chapter 5), runtime models (not limited to CzM and GMoDS) can be used to specify control interactions in HRI/HCI efficiently. Traditionally, an HRI/HCI is implemented as code that is scattered throughout the system. However, because runtime models depict a conceptual view that is easy for humans to understand, if a system reasons over runtime models, then humans can exert control over a system by manipulating the runtime models. Furthermore, the implementation of HRI/HCI would cleaner because the code would be located with the runtime models and not scattered throughout the system.

Bibliography

[1] Julie A. Adams. Supporting Human Supervision of Multiple Robots. *The Journal of the Robotics Society of Japan*, 24(5):17–19, July 2006.

[2] Srinivas M. Aji and Robert J. McEliece. The Generalized Distributive Law. *IEEE Transactions on Information Theory*, 46(2):325–343, March 2000.

[3] Laurel Allender, Troy D. Kelley, Lucia Salvi, John Lockett, Donald B. Headley, David Promisel, Diane Mitchell, Celine Richer, and Theo Feng. Verification, Validation, and Accreditation of a Soldier-System Modeling Tool. *Human Factors and Ergonomics Society Annual Meeting Proceedings*, 39:1219–1223, 1995.

[4] Susan Archer, Mala Gosakan, Paul Shorter, and John F. Lockett III. New Capabilities of the Army's Maintenance Manpower Modeling Tool. *Journal of the International Test and Evaluation Association*, 26(1):19–26, 2005.

[5] Ronald C. Arkin. Reactive Control as a Substrate for Telerobotic Systems. In *IEEE Aerospace and Electronic Systems Magazine*, volume 6, issue 6, pages 24–31, June 1991.

[6] Tucker Balch and Ronald C. Arkin. Communication in Reactive Multiagent Robotic Systems. *Autonomous Robots*, 1(1):27–52, March 1994.

[7] Gordon Blair, Nelly Bencomo, and Robert B. France. Models@ run.time. *Computer*, 42:22–27, 2009.

[8] Peter M. Blau. Formal Organization: Dimensions of Analysis. *The American Journal of Sociology*, 63(1):58–69, July 1957.

[9] S. C. Botelho and R. Alami. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1234–1239. IEEE, 1999.

[10] David J. Bruemmer, Douglas A. Few, Ronald L. Boring, Julie L. Marble, Miles C. Walton, and Curtis W. Nielsen. Shared Understanding for Collaborative Control. In *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, volume 35, issue 4, pages 494–504, July 2005.

[11] Kathleen M. Carley and Les Gasser. Computational Organization Theory. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 299–330, Cambridge, MA, USA, 1999. MIT Press.

[12] Nicholas L. Cassimatis. *Polyscheme: A Cognitive Architecture for Integrating Multiple Representation and Inference Schemes.* PhD thesis, Massachusetts Institute of Technology, 2002.

[13] Jessie Y. C. Chen, Michael J. Barnes, and Zhihua Qu. RoboLeader: An Agent for Supervisory Control of Multiple Robots. In *Proceedings of the 5th ACM/IEEE International Conference on Human-Robot Interaction*, HRI '10, pages 81–82, New York, NY, USA, 2010. ACM.

[14] Han-Lim Choi, Luc Brunet, and Jonathan P. How. Consensus-Based Decentralized Auctions for Robust Task Allocation. *IEEE Transactions on Robotics*, 25(4):912–926, August 2009.

[15] Klaus Christoffersen and David D. Woods. How to Make Automated Systems Team

Players. In Eduardo Salas, editor, *Advances in Human Performance and Cognitive Engineering Research*, volume 2. JAI Press/Elsevier, 2002.

[16] L. Conway, R. A. Voltz, and M. W. Walker. Teleautonomous Systems: Projecting and Coordinating Intelligent Action at a Distance. In *IEEE Transactions on Robotics and Automation*, volume 6, issue 2, pages 146–158, April 1990.

[17] Jacob W. Crandall and M. L. Cummings. Developing Performance Metrics for the Supervisory Control of Multiple Robots. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*, HRI '07, pages 33–40, New York, NY, USA, 2007. ACM.

[18] Jacob W. Crandall, Michael A. Goodrich, Dan R. Olsen Jr., and Curtis W. Nielsen. Validating Human-Robot Interaction Schemes in Multitasking Environments. In *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, volume 35, issue 4, pages 438–449, July 2005.

[19] Torbjørn S. Dahla, Maja Matarić, and Gaurav S. Sukhatme. Multi-robot task allocation through vacancy chain scheduling. *Robotics and Autonomous Systems*, 57(6–7):674–687, 2009.

[20] Raffaello D'Andrea and Michael Babish. The RoboFlag Testbed. In *Proceedings of the 2003 American Control Conference*, volume 1, pages 656–660, 2003.

[21] Rajdeep K. Dash, Perukrishnen Vytelingum, Alex Rogers, Esther David, and Nicholas R. Jennings. Market-Based Task Allocation Mechanisms for Limited-Capacity Suppliers. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 37(3):391–405, May 2007.

[22] Scott A. DeLoach. Modeling Organizational Rules in the Multi-agent Systems Engineering Methodology. In R. Cohen and B. Spencer, editors, *Advances in Artificial*

Intelligence: 15th Conference of the Canadian Society for Computational Studies of Intelligence, volume 2338 of *LNAI*, pages 1–15. Springer-Verlag, 2002.

[23] Scott A. DeLoach. Organizational Model for Adaptive Complex Systems. In Virginia Dignum, editor, *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. IGI Global, 2009.

[24] Scott A. DeLoach and Juan Carlos García-Ojeda. O-MaSE: A Customisable Approach to Designing and Building Complex, Adaptive Multi-agent Systems. *International Journal of Agent-Oriented Software Engineering*, 4(3):244–280, 2010.

[25] Scott A. DeLoach and Matthew Miller. A Goal Model for Adaptive Complex Systems. *International Journal of Computational Intelligence: Theory and Practice*, 5(2):83–92, 2010.

[26] Scott A. DeLoach, Walamitien Oyenan, and Eric T. Matson. A Capabilities-based Model for Adaptive Organizations. *Journal of Autonomous Agents and Multi-Agent Systems*, 16(1):13–56, February 2008.

[27] Scott A. DeLoach and Jorge L. Valenzuela. An Agent-Environment Interaction Model. In Lin Padgham and Franco Zambonelli, editors, *Agent-Oriented Software Engineering VII*, volume 4405 of *LNCS*, pages 1–18, Hakodate, Japan, August 2007. Springer Berlin / Heidelberg.

[28] Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman. Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering*, 1(2):231–258, 2001.

[29] M. Bernardine Dias. *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. PhD thesis, Carnegie Mellon University, 2004.

[30] Virginia Dignum. *A Model for Organizational Interaction: Based on Agents, Founded in Logic*. PhD thesis, Utrecht University, 2004.

[31] Virginia Dignum, Javier Vázquez-Salceda, and Frank Dignum. OMNI: Introducing Social Structure, Norms and Ontologies into Agent Organizations. In *PROMAS*, pages 181–198, 2004.

[32] X.C. Ding, M. Powers, M. Egerstedt, and R. Young. An Optimal Timing Approach to Controlling Multiple UAVs. In *American Control Conference*, pages 5374–5379, June 2009.

[33] Xu Chu Ding, Matthew Powers, Magnus Egerstedt, Shih-Yih (Ryan) Young, and Tucker Balch. Executive Decision Support: Single-Agent Control of Multiple UAVs. *IEEE Robotics Automation Magazine*, 16(2):73–81, June 2009.

[34] Marco Dorigo, Mauro Birattari, and Thomas Stützle. Ant Colony Optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, November 2006.

[35] Mica R. Endsley. Theoretical Underpinnings of Situation Awareness: A Critical Review. In Mica R. Endsley and Daniel J. Garland, editors, *Situation Awareness Analysis and Measurement*. Lawrence Erlbaum Associates, 2000.

[36] Terrence Fong, Chuck Thorpe, and Charles Baur. A Safeguarded Teleoperation Controller. In *IEEE International Conference on Advanced Robotics 2001*, August 2001.

[37] Cheng-Heng Fua and Shuzhi Sam Ge. COBOS: Cooperative Backoff Adaptive Scheme for Multirobot Task Allocation. In *IEEE Transactions on Robotics*, volume 21, issue 6, pages 1168–1178. IEEE, 2005.

[38] Juan C. García-Ojeda, Scott A. DeLoach, and Robby. agentTool III: From Process Definition to Code Generation. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, volume 2 of *AAMAS '09*, pages 1393–1394, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[39] Juan C. García-Ojeda, Scott A. DeLoach, and Robby. agentTool Process Editor: Supporting the Design of Tailored Agent-based Processes . In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 707–714, New York, NY, USA, 2009. ACM.

[40] Juan C. Garcia-Ojeda, Scott A. DeLoach, Robby, Walamitien H. Oyenan, and Jorge Valenzuela. O-MaSE: A Customizable Approach to Developing Multiagent Development Processes. In *8th International Workshop on Agent Oriented Software Engineering*, Honolulu, Hawaii, May 2007.

[41] Brian P. Gerkey and Maja J. Matarić. Sold!: Auction Methods for Multirobot Coordination. In *IEEE Transactions on Robotics and Automation*, volume 18, issue 5, pages 758–768. IEEE, 2002.

[42] Brian P. Gerkey and Maja J. Matarić. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *The International Journal of Robotics Research*, 23(9):939–954, September 2004.

[43] Michael A. Goodrich and Erwin R. Boer. Designing Human-Centered Automation: Tradeoffs in Collision Avoidance System Design. In *IEEE Transactions on Intelligent Transportation Systems*, volume 1, issue 1, pages 40–54, March 2000.

[44] Michael A. Goodrich, Timothy W. McLain, Jeffrey D. Anderson, Jisang Sun, and Jacob W. Crandall. Managing Autonomy in Robot Teams: Observations from Four

Experiments. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*, HRI '07, pages 25–32, New York, NY, USA, 2007. ACM.

[45] Object Management Group. Software & Systems Process Engineering Meta-Model Specification. `http://www.omg.org/spec/SPEM/2.0/`, 2008.

[46] Stephen B. Hughes and Michael Lewis. Task-Driven Camera Operations for Robotic Exploration. In *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, volume 35, issue 4, pages 513–522, July 2005.

[47] Matthew Johnson, Jeffrey M. Bradshaw, Paul J. Feltovich, Catholijn Jonker, Maarten Sierhuis, and Birna van Riemsdijk. Toward Coactivity. In *Proceeding of the 5th ACM/IEEE International Conference on Human-Robot Interaction*, pages 101–102, New York, NY, USA, 2010. ACM.

[48] Arjun Kumar Kanduri, Geb Thomas, Nathalie Cabrol, Edmond Grin, and Robert C. Anderson. The (In)Accuracy of Novice Rover Operators' Perception of Obstacle Height From Monoscopic Images. In *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, volume 35, issue 4, pages 505–512, July 2005.

[49] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, January 2003.

[50] Zsolt Kira and Mitchell A. Potter. Exerting Human Control Over Decentralized Robot Swarms. In *Proceedings of the 4th International Conference on Autonomous Robots and Agents*, pages 566–571, February 2009.

[51] Eric Krotkov, Reid Simmons, Fabio Cozman, and Sven Koenig. Safeguarded Teleoperation for Lunar Rovers: From Human Factors to Field Trials. In *IEEE Planetary Rover Technology and Systems Workshop*, 1996.

[52] C. Ronald Kube and Hong Zhang. Collective Robotics: From Social Insects to Robots. *Adaptive Behavior*, 2(2):189–218, September 1993.

[53] Masao Kubo and Yukinori Kakazu. Learning Coordinated Motions in a Competition for Food between Ant Colonies. In *Proceedings of the Third International Conference on Simulation of Adpative Behavior: From Animals to Animats 3*, pages 487–492, Cambridge, MA, USA, 1994. MIT Press.

[54] Michael Lewis, Jijun Wang, Stephen Hughes, and Xiong Liu. Experiments with Attitude: Attitude Displays for Teleoperation. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 2, pages 1345–1349, October 2003.

[55] Jane W. S. Liu. *Real-Time Systems*. Prentice Hall, Upper Saddle River, NJ, USA, 1st edition, 2000.

[56] M2 Technologies. *M2 and MC IED Awareness: Controlling Robots Teams in Urban Environments*.

[57] Kathryn S. Macarthur, Ruben Stranders, Sarvapali D. Ramchurn, and Nicholas R. Jennings. A Distributed Anytime Algorithm for Dynamic Task Allocation in Multi-Agent Systems. In *Twenty-Fifth Conference on Artificial Intelligence*. AAAI Press, August 2011.

[58] Maja J. Matarić. Designing Emergent Behaviors: From Local Interactions to Collective Intelligence. In *Proceedings of the Third International Conference on Simulation of Adpative Behavior: From Animals to Animats 2*, pages 432–441, Cambridge, MA, USA, 1993. MIT Press.

[59] Sandra Mau and John Dolan. Scheduling for Humans in Multirobot Supervisory Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1637–1643, November 2007.

[60] James McLurkin and Jennifer Smith. Distributed Algorithms for Dispersion in Indoor Environments Using a Swarm of Autonomous Mobile Robots. In Rachid Alami, Raja Chatila, and Hajime Asama, editors, *7th International Symposium on Distributed Autonomous Robotic Systems*. Springer, 2004.

[61] James McLurkin, Jennifer Smith, James Frankel, David Sotkowitz, David Blau, and Brian Schmidt. Speaking Swarmish: Human-Robot Interface Design for Large Swarms of Autonomous Mobile Robots. In *To Boldly Go Where No Human-Robot Team Has Gone Before*, AAAI Spring Symposium, pages 72–75, Menlo Park, California, 2006. AAAI Press.

[62] François Michaud, Jean-François Laplante, Hélène Larouche, Audrey Duquette, Serge Caron, Dominic Létourneau, and Patrice Masson. Autonomous Spherical Mobile Robot for Child-Development Studies. In *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, volume 35, issue 4, pages 471–480, July 2005.

[63] Christopher A. Miller, Harry B. Funk, Michael Dorneich, and Stephen D. Whitlow. A Playbook Interface for Mixed Initiative Control of Multiple Unmanned Vehicle Teams. In *Proceedings of the 21st Digital Avionics Systems Conference*, volume 2 of *21st DASC*, pages 7E4–1–7E4–13, Irvine, CA, USA, 2002.

[64] Christopher A. Miller and Raja Parasuraman. Designing for Flexible Interaction Between Humans and Automation: Delegation Interfaces for Supervisory Control. *Human Factors*, 49(1):57–75, 2007.

[65] Christopher A. Miller, Michael Pelican, and Robert Goldman. "Tasking" Interfaces for Flexible Interaction with Automation: Keeping the Operator in Control. In

Proceedings of the Conference on Human Interaction with Complex Systems, pages 188–202, 2000.

[66] Matthew Miller. A Goal Model for Dynamic Systems. Master's thesis, Kansas State University, April 2007.

[67] Yashodhan Nevatia, Todor Stoyanov, Ravi Rathnam, Max Pfingsthorn, Stefan Markov, Rares Ambrus, and Andreas Birk. Augmented Autonomy: Improving human-robot team performance in Urban Search and Rescue. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2103–2108, September 2008.

[68] Donald A. Norman. *The Design of Everyday Things*. Doubleday/Currency, 1988.

[69] Raja Parasuraman, Scott Galster, Peter Squire, Hiroshi Furukawa, and Christopher Miller. A Flexible Delegation-Type Interface Enhances System Performance in Human Supervision of Multiple Robots: Empirical Studies With RoboFlag. In *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, volume 35, issue 4, pages 481–493, July 2005.

[70] Raja Parasuraman, Thomas B. Sheridan, and Christopher D. Wickens. A Model for Types and Levels of Human Interaction with Automation. In *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, volume 30, issue 3, pages 286–297, May 2000.

[71] Lynne E. Parker. ALLIANCE: An Architecture for Fault Tolerant Multirobot Cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

[72] Lynne E. Parker. Distributed Intelligence: Overview of the Field and its Application in Multi-Robot Systems. *Journal of Physical Agents*, 2(1):5–14, 2008.

[73] Kevin M. Passino. Biomimicry of Bacterial Foraging for Distributed Optimization and Control. *IEEE Control Systems Magazine*, 22(3):52–67, 2002.

[74] Nuria Pelechano, Kevin O'Brien, Barry Silverman, and Norman Badler. Crowd Simulation Incorporating Agent Psychological Models, Roles and Communication. In *First International Workshop on Crowd Simulation*, pages 21–30, 2005.

[75] David Pietrocola and Barry G. Silverman. Taxonomy and Method for Handling Large and Diverse Sets of Interactive Objects in Immersive Environments. In *Proceedings of the 19th Conference on Behavior Representation in Modeling and Simulation*, pages 256–262, March 2010.

[76] Gregg Podnar, John Dolan, Alberto Elfes, and Marcel Bergerman. Multi-Level Autonomy Robot Telesupervision. *Robotics Institute*, Paper(181), 2008.

[77] Sarvapali D. Ramchurn, Alessandro Farinelli, Kathryn S. Macarthur, and Nicholas R. Jennings. Decentralized Coordination in RoboCup Rescue. *The Computer Journal*, 53(9):1447–1461, 2010.

[78] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[79] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2002.

[80] Paul Scerri, Elizabeth Liao, Justin Lai, Katia Sycara, Yang Xu, and Mike Lewis. Coordinating Very Large Groups of Wide Area Search Munitions. *Theory and Algorithms for Cooperative Systems*, 2005.

[81] Douglas C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, February 2006.

[82] Jean Scholtz. Theory and Evaluation of Human Robot Interactions. In *Proceedings of the 36th Hawaii International Conference on System Sciences*, 2003.

[83] Jean C. Scholtz, Brian Antonishek, and Jeff D. Young. Implementing of a Situation Awareness Assessment Tool for Evaluation of Human-Robot Interfaces. In *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, volume 35, issue 4, pages 450–459, July 2005.

[84] Barry G. Silverman. Performance Moderator Functions for Human Behavior Modeling in Military Simulations. `http://www.seas.upenn.edu/~barryg/PMFset.zip`, 2002.

[85] Barry G. Silverman. Toward Realism in Human Performance Simulation. In James W. Ness, Darren R. Ritzer, and Victoria Tepe, editors, *The Science and Simulation of Human Performance*, volume 5, pages 469–498. Emerald Group Publishing, 2004.

[86] Barry G. Silverman, Michael Johns, Jason Cornwell, and Kevin O'Brien. Human Behavior Models for Agents in Simulators and Games: Part I: Enabling Science with PMFserv. *Presence: Teleoperators and Virtual Environments*, 15(2):139–162, 2006.

[87] Barry G. Silverman, Kevin O'Brien, Jason Cornwell, Michael Johns, and Gnana Bharathy. Mathematical Theory of Software Agent Behavior and Human Behavior Models (HBMs) to Increase the Realism of Synthetic Agents. ONR N00014-02-1-0621, Ackoff Center for Advancement of Systems Approaches (ACASA), 2003.

[88] Barry G. Silverman, David Pietrocola, Nathan Weyer, Ransom Weaver, Nouva Esomar, Robert Might, and Deepthi Chandrasekaran. NonKin Village: An Embeddable Training Game Generator for Learning Cultural Terrain and Sustainable Counter-Insurgent Operations. In Frank Dignum, Jeff Bradshaw, Barry Silverman, and Willem van Doesburg, editors, *Agents for Games and Simulations*, volume 5920

of *Lecture Notes in Computer Science*, pages 135–154. Springer Berlin / Heidelberg, 2009.

[89] Reid Simmons, Sanjiv Singh, David Hershberger, Josue Ramos, and Trey Smith. First Results in the Coordination of Heterogeneous Robots for Large-Scale Assembly. In *Experimental Robotics VII*, volume 271 of *Lecture Notes in Control and Information Sciences*, pages 323–332. Springer Berlin / Heidelberg, 2001.

[90] Mark G. Snyder, Zhihua Qu, Jessie Y.C. Chen, and Michael J. Barnes. RoboLeader for Reconnaissance by a Team of Robotic Vehicles. In *International Symposium on Collaborative Technologies and Systems (CTS)*, pages 522–530, May 2010.

[91] Daniel J. Stilwell and John S. Bay. Toward the Development of a Material Transport System using Swarms of Ant-like Robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 766–771, Atlanta, GA, USA, 1993.

[92] Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, June 1999.

[93] Kazuo Sugihara and Ichiro Suzuki. Distributed Algorithms for Formation of Geometric Patterns with Many Mobile Robots. *Journal of Robotic Systems*, 13(3):127–139, 1996.

[94] Sang-Joon Sun, Dong-Wook Lee, and Kwee-Bo Sim. Artificial Immune-Based Swarm Behaviors of Distributed Autonomous Robotic Systems. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 4, pages 3993–3998, 2001.

[95] Fang Tang and Lynne E. Parker. ASyMTRe: Automated Synthesis of Multi-Robot Task Solutions through Software Reconfiguration. In *Proceedings of the 2005 IEEE*

International Conference on Robotics and Automation (ICRA), pages 1501–1508. IEEE, April 2005.

[96] Fang Tang and Lynne E. Parker. Distributed multi-robot coalitions through asymtre-d. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2606–2613. IEEE, August 2005.

[97] J. Gregory Trafton, Nicholas L. Cassimatis, Magdalena D. Bugajska, Derek P. Brock, Farilee E. Mintz, and Alan C. Schultz. Enabling Effective Human-Robot Interaction Using Perspective-Taking in Robots. In *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, volume 35, issue 4, pages 460–470, July 2005.

[98] Athanasios Tsalatsanis, Ali Yalcin, and Kimon P. Valavanis. Optimized Task Allocation in Cooperative Robot Teams. *17th Mediterranean Conference on Control and Automation*, pages 270–275, 2009.

[99] Toshio Tsuji and Yoshiyuki Tanaka. Tracking Control Properties of Human-Robotic Systems Based on Impedance Control. In *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, volume 35, issue 4, pages 523–535, July 2005.

[100] Axel van Lamsweerde, Emmanual Letier, and Robert Darimont. Managing Conflicts in Goal-Driven Requirements Engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, 1998.

[101] Lovekesh Vig and Julie A. Adams. Multi-Robot Coalition Formation. In *IEEE Transactions on Robotics*, volume 22, issue 4, pages 637–649, August 2006.

[102] Lovekesh Vig and Julie A. Adams. Coalition Formation: From Software Agents to Robots. *Journal of Intelligent and Robotic Systems*, 50(1):85–118, September 2007.

[103] Regis Vincent, Dieter Fox, Jonathan Ko, Kurt Konolige, Benson Limketkai, Benoit Morisset, Charles Ortiz, Dirk Schulz, and Benjamin Stewart. Distributed multirobot exploration, mapping, and task allocation. *Annals of Mathematics and Artificial Intelligence*, 52(2–4):229–255, April 2008.

[104] Javier Vázquez-Salceda and Frank Dignum. Modelling Electronic Organizations. In Vladimir Marik, Jörg Müller, and Michal Pechoucek, editors, *Multi-Agent Systems and Applications III*, volume 2691 of *LNAI*, pages 584–593. Springer-Verlag, 2003.

[105] Huadong Wang, Andreas Kolling, Nathan Brooks, Michael Lewis, and Katia Sycara. Synchronous vs. Asynchronous Control for Large Robot Teams. In Randall Shumaker, editor, *Virtual and Mixed Reality - Systems and Applications*, volume 6774 of *Lecture Notes in Computer Science*, pages 415–424. Springer Berlin / Heidelberg, 2011.

[106] Huadong Wang, Andreas Kolling, Nathan Brooks, Sean Owens, Shafiq Abedin, Paul Scerri, Pei ju Lee, Shih-Yi Chien, Michael Lewis, and Katia Sycara. Scalable Target Detection for Large Robot Teams. In *Proceedings of the 6th International Conference on Human-Robot Interaction*, HRI '11, pages 363–370, New York, NY, USA, 2011. ACM.

[107] iRobot PackBot. `http://www.irobot.com/sp.cfm?pageid=171`.

[108] Robotics. `http://www.cbc.ca/news/background/tech/robotics/military.html`.

[109] Christopher D. Wickens, John D. Lee, Yili Liu, and Sallie E. Gordon-Becker. *An Introduction to Human Factors Engineering*. Prentice Hall, 2nd edition, November 2003.

[110] David D. Woods, James Tittle, Magnus Feil, and Axel Roesler. Envisioning Human-Robot Coordination in Future Operations. In *IEEE Transactions on Systems, Man*

& Cybernetics: Part C: Special Issue on Human-Robot Interaction, volume 34, issue 2, pages 210–218, May 2004.

[111] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):303–328, 2001.

[112] Yu Zhang and Lynne E. Parker. A General Information Quality Based Approach for Satisfying Sensor Constraints in Multirobot Tasks. In *IEEE International Conference on Robotics and Automation*, pages 1452–1459, May 2010.

[113] Yu Zhang and Lynne E. Parker. IQ-ASyMTRe: Synthesizing Coalition Formation and Execution for Tightly-Coupled Multirobot Tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5595–5602, October 2010.

[114] Kuanhao Zheng, Dylan F. Glas, Takayuki Kanda, Hiroshi Ishiguro, and Norihiro Hagita. How Many Social Robots Can One Operator Control? In *Proceedings of the 6th International Conference on Human-Robot Interaction*, HRI '11, pages 379–386, New York, NY, USA, 2011. ACM.

[115] Christopher Zhong. Cooperative Robotic Organization Simulator. `http://macr.cis.ksu.edu/cros`.

[116] Robert Zlot and Anthony Stentz. Market-based Multirobot Coordination for Complex Tasks. *The International Journal of Robotics Research*, 25(1):73–101, 2006.

Appendix A

Acronyms

ACO Ant Colony Optimization . . . 145

aT^3 agentTool III . . . 27

AuRA Autonomous Robot Architecture . . . 37

AV Autonomous Vehicle . . . 40

BAM Biological Agents Module . . . 16

CAAS Collision and Accident Avoidance System . . . 38

CC Control Component . . . 91

CMS Conference Management System . . . 54

CPR Cardiopulmonary Resuscitation . . . 88

CROS Cooperative Robotics Organization Simulator . . . 104

CzM Chazm Model . . . 21

DES Discrete Event System . . . 157

dSSPT double Shifted Shortest Processing Time . . . 167

EC Execution Component . . . 91

ERO Emergency Response Officer . . . 16

GMoDS Goal Model for Dynamic Systems . . . 21

GR Goal Reasoning . . . 92

HCI Human-Computer Interaction . . . 1

HPMF Human Performance Moderator Function . . . 10

HRI Human-Robot Interaction . . . 1

HuRT Human Robot Teams . . . 134

IAL Intelligent Autonomous Layer . . . 85

IED Improvised Explosive Device . . . 93

IMPRINT Improved Performance Research Integration Tool . . . 159

LED Light-Emitting Diode . . . 165

MAC Mission Analysis Component . . . 162

MACES Multi-Agent Communication for Evacuation Simulation 160

MARTA Multilevel-Autonomy Robot Telesupervision Architecture 166

MAS Multiagent System 6

MaSE Multiagent Systems Engineering 27

MDE Model Driven Engineering 11

NN Neural Network 41

OBAA Organization-Based Agent Architecture 90

OM Organization Model 92

OMACS Organization Model for Adaptive Computational Systems 7

OMAS Organization-based Multiagent System 7

O-MaSE Organization-based Multiagent System Engineering 21

OMNI Organizational Model for Normative Institutions 7

OperA Organizations per Agents 7

ORC Organizational Reasoning Component 92

PMF Performance Moderator Function 10

PMFserv Performance Moderator Functions Server 159

RA Reorganization Algorithm . . . 92

RCC Role Control Component . . . 91

SHAM Structural Hazards Awareness Module . . . 16

TOP Team Oriented Plan . . . 164

UAV Uninhabited Aerial Vehicle . . . 163

UCAV Uninhabited Combat Aerial Vehicle . . . 161

UML Unified Modeling Language . . . 206

USMC United States Marine Corp . . . 93

Appendix B

Emergency Response Team

The following three sections describe the three functions of the emergency response team and the responsibilities and actions of team members when carrying out their assigned function(s).

B.1 Survey Group

The survey group consists of several teams of human-robot pairs. The size of a team can grow or shrink to include multiple humans as the need arises but a typical team contains one human and one robot. The purpose of the survey group is to quickly survey the target area to provide an accurate and up-to-date overview of the layout so that the other two groups (hazard identification and victim rescue) of the emergency response team can respond quickly and appropriately.

The robots in the survey group are equipped with the following additional capabilities: (1) an infrared camera, which identifies heat signatures of survivors; (2) a standard camera, which detects motion, structural and environmental hazards, and performs image recognition of survivors; (3) a microphone, which detects audio cues of hazards and responses from

survivors; and (4) a speaker, which is used to verbally communicate to survivors.

The following is a list of actions taken by the robots when surveying a target area.

1. The robot decides with the human partner on how to get to the target area.

    - If the human partner decides to carry the robot to the target area, the robot goes into standby mode until arrival at the target area.

    - If the robot decides to lead the way to the target area, the robot plots a path to the target area. Once a path is plotted, the robot navigates the path to the target area. If there are path corrections from the human partner, the robot updates the path with the corrections.

2. Upon reaching the target area, the robot begins to systematically search the target area (attempting to reach 100% coverage).

    - The map for the target area is updated by the robot as the robot is searching the target area so that other members of the emergency response team can utilize the updated map.

    - If the robot encounters an area that it cannot navigate, the robot asks the human partner to take over and search the area while the robot resumes its systematic search of the target area.

    - If the robot encounters structural or environmental hazards, the robot notes the location of each hazard on the map and notifies the commander.

    - If readings from robot's sensors indicate a potential victim, the robot attempts to confirm the readings.

        (a) The robot attempts to move closer to the source of the readings. If the robot cannot get closer to verify the readings, the robot asks the human partner to take over and the robot resumes its systematic search.

(b) Once the robot is close enough to the source, the robot verifies the initial readings. If the readings are consistent with a victim, the robot determines the status of the victim: survivor (conscious or unconscious), or deceased.

  i. The robot plays a prerecorded message through the speaker and waits for a response from the victim.

  ii. The robot attempts to capture a response from the victim either verbally or visually.

(c) Once the robot has ascertained the status of the victim, the robot notifies the human partner of victim. If the human partner is already occupied with another victim, the robot notifies the commander for additional help and waits for the (replacement) human partner to arrive.

(d) Once the human partner arrives at the robot's location, the robot resumes its systematic search.

The following is a list of actions taken by the humans when surveying a target area.

1. The human decides with the robot partner on how to get to the target area.

   - If the human decides to carry the robot partner to the target area, the human waits for the robot partner to enter standby mode. Once the robot partner is in standby mode, the human straps the robot partner onto the backpack and proceeds to the target area.

   - If the robot partner decides to lead the way to the target area, the human follows the robot partner to the target area. If the human spots a better path while following the robot, the human notifies the robot partner of corrections to the path.

2. Once the human arrives at the target area, the human activates the robot (if it is

in standby mode) and performs a visual inspection of the target area for potential victims and hazards.

- The human updates the maps for the target area along with the robot partner. The human could also make corrections to the updates made by the robot partner.
- If the human receives a request from the robot partner to search an area, the human takes over and proceeds to the indicated area and begins to search the area.
- If the human receives a request to investigate a potential victim, the human takes over and proceeds to the indicated location and verifies that there is a potential victim. If there is indeed a victim, the human verifies the status of the victim: survivor (conscious or unconscious and mobile or immobile), or deceased.
  - If the victim is deceased, the human marks the location of body with a flag and notifies the commander.
  - If the survivor is mobile, the human notifies the commander of a survivor and escorts the survivor back to base.
  - If the survivor is immobile, the human notifies the commander and stays with survivor until the rescue team arrives.
- If the human receives a notification from the robot partner about the discovery of a victim, the human proceeds to victim's location.
  - If the victim is deceased, the human marks the location of body with a flag and notifies the commander.
  - If the survivor is mobile, the human notifies the commander of a survivor and escorts the survivor back to base.
  - If the survivor is immobile, the human notifies the commander and stays with survivor until the rescue team arrives.

B.2 Hazard Identification Group

When the survey group notes any structural or environmental hazards, the commander dispatches teams to investigate the hazards. The teams are responsible for identifying the nature of the hazards and for making recommendations to the commander whether specialized teams such as the bomb squad, chemical squad, or radiation squad need to be called in to handle the hazards.

The robots in the hazard identification group are equipped with the following additional capabilities: (1) a standard camera, which is used for detecting hazards; (2) a secondary camera that has full range of motion, which allows the human partner to visually inspect areas; (3) a variety of temperature sensors; (4) a variety of chemical sensors such as electrochemical, pellistor, infrared, and thermal conductivity, which detects toxic chemicals, flammables, and carbon dioxide; (5) a variety of instruments for collecting samples; (6) a variety of chemical analyzers.

The following is a list of actions taken by the robot when identifying a hazard.

1. The robot decides with the human partner on how to get to the target location.

    - If the human partner decides to carry the robot to the target location, the robot goes into standby mode until arrival in the target location.

    - If the robot decides to lead the way to the target location, the robot plots a path to the target location. Once a path is plotted, the robot navigates the path to the target location. If there are path corrections from the human partner, the robot updates the path with the corrections.

2. Upon arrival at the target location, the robot decides with the human partner on which areas to look for the hazards.

3. Once the areas have been decided, the robot starts to search the areas for hazards

using its sensors.

- If the robot detects toxic chemicals, the robot notifies the human partner. The robot analyzes the toxic chemicals and makes a recommendation to the human partner: (1) safe with protective gear on or (2) unsafe with protective gear on. If the human partner decides to stay at a distance, the robot takes on the areas that have not been searched by the human partner.
- If the robot detects a sudden temperature change, the robot notifies the human partner and takes over the searching the areas that have not been searched by the human partner.
- If the robot detects a hazard, the robot notifies the human partner.
  (a) Once the human partner arrives, the robot proceeds with the identification process.
  (b) The robot uses the camera to identify various spots where samples are to be collected. The robot coordinates with the human partner to avoid collecting multiple samples from the same area. If there are areas where the robot cannot reach to collect the samples, the robot notifies the human partner of the areas and lets the human partner collect the samples instead.
  (c) The robot proceeds to collect samples from reachable areas and places the samples into test tubes for analysis.
  (d) Once the samples are in test tubes, the robot proceeds to analyze the samples.
  (e) Once the analysis of the samples are completed, the robot confers with the human partner on the next course of action (such as whether to call in specialized squads) and makes a recommendation to the commander.

The following is a list of actions taken by the humans when identifying a hazard.

1. The human decides with the robot partner on how to get to the target location.

    - If the human decides to carry the robot partner to the target location, the human waits for the robot partner to enter standby mode. Once the robot partner is in standby mode, the human straps the robot partner onto the backpack and proceeds to the target location.

    - If the robot partner decides to lead the way to the target location, the human follows the robot partner to the target location. If the human spots a better path while following the robot, the human notifies the robot partner of corrections to the path.

2. Upon arrival at the target location, the human decides with the robot partner on which areas to look for the hazards.

3. Once the areas have been decided, the human begins to search the areas for hazards.

    - If the robot partner notifies the human of toxic chemicals, the human dons the protective head gear and waits for results of the analysis of the toxic chemicals from the robot partner. Based on the analysis results, the human decides whether to continue searching or to leave the searching to the robot. If the human decides to leave the searching to the robot, the human can utilize the secondary camera of the robot partner as a second set of eyes in the search process.

    - If the robot partner notifies the human of a sudden temperature change, the human evacuates the immediate area and monitors the situation using the secondary camera of the robot partner.

    - If the human notices a hazard, the human notifies the robot partner.

        (a) Once the robot partner arrives, the human proceeds with the identification process.

(b) The human determines areas where the samples are to be collected. The human coordinates with the robot partner to avoid collecting multiple samples from the same area.

(c) The human proceeds to collect samples from areas that the robot partner cannot reach and hands the test tubes to the robot partner for analysis.

(d) Once the analysis of the samples are complete, the human discusses the results with the robot partner and decides on the next course of action such as whether to call in specialized squads.

B.3 Victim Rescue Group

When the survey group notes any victims, the commander dispatches teams to rescue the victims. The teams are responsible for vacating victims from the scene to a safer area to receive further medical assistance.

The robots in the victim rescue group are equipped with a stretcher, which allows the robot to carry a victim back to safety.

The following is a list of actions taken by the robot when rescuing a victim.

1. The robot plots a path to the target location. Once a path is plotted, the robot navigates along path to the target location. If there are path corrections from the human partner, the robot updates the path with the corrections.

2. Once the robot arrives at the survivor's location, the robot prepares the stretcher for the survivor.

3. The robot waits for the survivor to be secured to the stretcher.

4. Once the survivor is secured to the stretcher, the robot proceeds back to base with the human partner.

The following is a list of actions taken by the human when rescuing a victim.

1. The human follows the robot partner to the target location. If the human spots a better path while following the robot, the human notifies the robot partner of corrections to the path.

2. Once the human arrives at the survivor's location, the human collaborates with the other human on how to secure the survivor to the stretcher.

3. Once the survivor is secured to the stretcher, the human notifies the robot partner and proceeds back to base with the robot partner.

Appendix C

Creating Runtime Models

Based on a book by Norman [68], a conceptual model (also known as a mental model) is a simplification about how a system works, in which the system is broken down into various parts and the relationship between the parts. Humans use a conceptual model to (1) understand how a system works; (2) should problems arise, reason about the model to derive solutions; and (3) make decisions based on information in the model.

A runtime model is also a conceptual model. However, not all conceptual models can be runtime models. A runtime model is an implementation of a conceptual model that exists in the memory of a computing system. The system reasons over the runtime model to make decisions.

In order to develop a runtime model from a conceptual model, the conceptual model should have the following two properties: (1) the conceptual model can be fully represented as a connected graph (a disconnected graph is basically multiple conceptual models lumped together with no apparent relationships between the models) and (2) the conceptual model should capture meaningful information pertaining to the associated domain so that problems can be identified and addressed autonomously.

A graph consists of two elements: nodes and edges. A node corresponds to an entity in

the conceptual model such as the *role* or *goal* entity. An edge corresponds to the relationship between the entities such as a role *achieves* a goal. Furthermore, the type of relationship should be explicitly specified. There are four types of relationships: (1) one-to-one, (2) one-to-many, (3) many-to-one, and (4) many-to-many.

Once the conceptual model is fully represented as a connected graph and contains the relationship-types, the graph can be translated to a Unified Modeling Language (UML) class diagram, where the nodes correspond to classes and the edges correspond to associations. There are multiple possibilities for capturing edges. For example, an edge can be captured as a class that contains two pointers to the two other classes. Certain implementations may be more efficient for some uses. However, there is typically no single implementation that is most efficient for all uses.

Once the translation is completed, attributes and method can be defined in the UML class diagram for implementation. The process for implementing a UML class diagram is the standard process and is not covered in this dissertation.

Appendix D

Time Complexity Analysis

D.1 Random Algorithm

Figure D.1 shows the pseudo code for the random algorithm. The random algorithm computes a set of goals that has not been assigned yet (g') from the given set of goals (W_g) and it iterates through every goal from the set of unassigned goals (g'). From a goal (g_i), the set of roles (W_r) that can achieve g_i is obtained via the `achieves()` function. The set of agents (W_a) is copied to a' and an agent (a_i) is randomly picked from a'; the `random()` function randomly picks an element without replacement (i.e., a_i is removed from a'). An iteration occurs over every role (r_i) to determine viable assignments (α), which are computed using the `goodness()` function. Once α is computed, an assignment (α_i) is randomly selected. If there are no viable assignments for a_i, another agent is randomly picked to repeat the process until either a viable assignment is found or all agents have been picked and there are no viable assignments for g_i.

Proof. The time complexity of the random algorithm is $O(g \times a \times r \times c)$.

Let g be the number of goals for the input W_g, a be the number of agents for the input W_a, r be the number of roles in the organization, and c be the number of capabilities in the

```
function reorganize(W_g, W_a)
  g' ← unassigned(W_g), β ← ∅
  for each g_i from g' do
    a' ← W_a, W_r ← achieves(g_i)
    repeat
      a_i ← random(a'), α ← ∅
      for each r_i from W_r do
        if goodness(r_i, a_i, g_i) > 0 then
          α ← α ∪ ⟨r_i, a_i, g_i⟩
        end if
      end for
      α_i ← random(α)
    until α_i ≠ ∅ or a' = ∅
    β ← β ∪ α_i
  end for
  return β
```

Figure D.1: Pseudo Code – Random Algorithm

organization.

Computing the set of unassigned goals requires checking each goal to determine if it is assigned. Thus, the `unassigned()` function takes $\Theta(g)$ time.

The first loop iterates through all unassigned goals. In the worst case, all goals from W_g are not assigned and the loop takes $O(g)$ time. The time complexity so far is $O(g + g)$.

Duplicating the set of agents takes $\Theta(a)$ time because it iterates through each agent and since this duplication occurs inside the first loop, the time complexity so far is $O(g+(g \times a))$.

The `achieves()` function, which returns a set of roles (W_r), takes constant time, although the cardinality of W_r varies. In the best case, every goal is achieved by 1 role; however, in the worst case, every goal can be achieved by r roles. The time complexity remains unchanged.

The second loop terminates when either a viable assignment is found or when a' is $\emptyset$. Randomly picking an agent through the `random()` function takes constant time. In the best case, the first randomly picked agent is capable ($\Theta(1)$). In the worst case, no agents are

capable or the only capable agent is the last one to be picked ($\Theta(a)$). So, the time it takes for this loop is $O(a)$. Since the second loop occurs inside the first loop, the time complexity so far is $O(g + (g \times (a + a)))$.

The third loop iterates through every role from W_r, which has a worst case cardinality of r. Thus, the loop takes $O(r)$. Since the third loops occurs inside the second loop, the time complexity so far is $O(g + (g \times (a + (a \times r))))$.

The `goodness()` function takes $O(c)$ because every capability required by the role has to be checked, which is c in the worst case as all the capabilities in the organization could be required by a role. Since the `goodness()` function is called inside the third loop, the time complexity so far is $O(g + (g \times (a + (a \times (r \times c)))))$.

Since the remaining pseudo code is constant time, the time complexity of the random algorithm is $O(g + g \times (a + a \times r \times c))$ and can be simplified to $O(g \times a \times r \times c)$. □

D.2 Round Robin Algorithm

Figure D.2 shows the pseudo code for the round robin algorithm. The round robin algorithm starts by computing the set of unassigned goals (g') from the given goals (W_g). In addition, it also sorts the unassigned goals so *review goals* for the same paper are grouped together. Next, the algorithm iterates through each goal (g_i) and obtains the roles (W_r) that can achieve g_i. It then iterates through each agent (a_i) to determine the best role for the a_i and g_i, which requires iterating through each role (r_i) and selecting the role with the best `goodness` score. If a_i is capable (i.e., $\beta \neq \emptyset$), the number of assignments (including those that the algorithm has already decided to assign) for a_i is obtained from the `assignments()` function. If the previously stored result (γ) is $\emptyset$ or the number of assignments is less than the one from γ, this agent (a_i) becomes the one that will be assigned to the goal (g_i). Otherwise, the algorithm moves to the next agent and repeats the process until all agents have been

checked. The agent ($\gamma.a$) with least number of assignments is assigned to the goal (g_i). The algorithm moves to the next goal and repeats the process until all goals are processed.

```
function reorganize(W_g, W_a)
  g' ← sort(unassigned(W_g)), λ ← ∅
  for each g_i from g' do
    W_r ← achieves(g_i), γ ← ∅
    for each a_i from W_a do
      β ← ∅
      for each r_i from W_r do
        α ← goodness(r_i, a_i, g_i)
        if α > 0 and (β = ∅ or α > β.α) then
          β ← ⟨r_i, a_i, g_i, α⟩
        end if
      end for
      if β ≠ ∅ then
        δ ← assignments(β.a)
        if γ = ∅ or δ < γ.δ then
          γ ← ⟨β.r, β.a, β.g, δ⟩
        end if
      end if
    end for
    λ ← λ ∪ ⟨γ.r, γ.a, γ.g⟩
  end for
  return λ
```

Figure D.2: Pseudo Code – Round Robin Algorithm

Proof. The time complexity of the round robin algorithm is $O(g \times a \times r \times c)$.

Let g be the number of goals for the input W_g, a be the number of agents for the input W_a, r be the number of roles in the organization, and c be the number of capabilities in the organization.

Computing the set of unassigned goals requires checking each goal to determine if it is assigned. Thus, the `unassigned()` function takes $\Theta(g)$ time. Next, the unassigned goals are sorted, which takes $O(g \times \log g)$. The time complexity so far is $O(g + (g \times \log g))$.

The first loop iterates through all unassigned goals. In the worst case, all goals from W_g

are not assigned and the loop takes $O(g)$ time. The time complexity so far is $O(g + (g \times \log g) + g)$.

The `achieves()` function, which returns a set of roles (W_r), takes constant time, although the cardinality of W_r varies. In the best case, every goal is achieved by 1 role; however, in the worst case, every goal can be achieved by r roles. The time complexity remains unchanged.

The second loop iterates through every agent from W_a and thus, the time complexity for this loop is $\Theta(a)$. Since the second loop occurs inside the first loop, the time complexity so far is $O(g + (g \times \log g) + (g \times a))$.

The third loop iterates through every role from W_r, which has a worst case cardinality of r. Thus, the loop takes $O(r)$. Since the third loops occurs inside the second loop, the time complexity so far is $O(g + (g \times \log g) + (g \times (a \times r)))$.

The `goodness()` function takes $O(c)$ because every capability required by the role has to be checked, which is c in the worst case as all the capabilities in the organization could be required by a role. Since the `goodness()` function is called inside the third loop, the time complexity so far is $O(g + (g \times \log g) + (g \times (a \times (r \times c))))$.

Since the remaining pseudo code is constant time, the time complexity of the round robin algorithm is $O(g + g \times \log g + g \times a \times r \times c)$ and can be simplified to $O(g \times a \times r \times c)$. $\square$

D.3 Greedy Algorithm

Figure D.3 show the pseudo code for the greedy algorithm. The greedy algorithm starts by computing the set of unassigned goals (g') from the given goals (W_g). Next, the algorithm iterates through each goal (g_i) and obtains the roles (W_r) that can achieve the g_i. It then iterates through each agent (a_i) to determine the best role for the a_i and g_i, which requires iterating through each role (r_i) and selecting the role with the best `goodness` score. If a_i is

capable (i.e., $\beta \neq \emptyset$), the δ is compared to the δ of the previously saved result (γ). If γ is $\emptyset$ or the δ is greater than the δ of γ, this agent (a_i) becomes the one that will be assigned to the goal (g_i). The algorithm moves to the next agent and repeats the process until all agents have been checked. The agent ($\gamma.a$) with the highest score is assigned to the goal (g_i). The algorithm moves to the next goal and repeats the process until all goals are processed.

```
function reorganize(W_g, W_a)
  g' ← unassigned(W_g), δ ← ∅
  for each g_i from g' do
    W_r ← achieves(g_i), γ ← ∅
    for each a_i from W_a do
      β ← ∅
      for each r_i from W_r do
        α ← goodness(r_i, a_i, g_i)
        if α > 0 and (β = ∅ or α > β.α) then
          β ← ⟨r_i, a_i, g_i, α⟩
        end if
      end for
      if β ≠ ∅ then
        δ ← β.α ÷ assignments(β.a)
        if γ = ∅ or δ > γ.δ then
          γ ← ⟨β.r, β.a, β.g, β.δ⟩
        end if
      end if
    end for
    δ ← δ ∪ ⟨γ.r, γ.a, γ.g⟩
  end for
  return δ
```

Figure D.3: Pseudo Code – Greedy Algorithm

Proof. The time complexity of the greedy algorithm is $O(g \times a \times r \times c)$.

Let g be the number of goals for the input W_g, a be the number of agents for the input W_a, r be the number of roles in the organization, and c be the number of capabilities in the organization.

Computing the set of unassigned goals requires checking each goal to determine if it is

assigned. Thus, the `unassigned()` function takes $\Theta(g)$ time.

The first loop iterates through all unassigned goals. In the worst case, all goals from W_g are not assigned and the loop takes $O(g)$ time. The time complexity so far is $O(g+g)$.

The `achieves()` function, which returns a set of roles (W_r), takes constant time, although the cardinality of W_r varies. In the best case, every goal is achieved by 1 role; however, in the worst case, every goal can be achieved by r roles. The time complexity remains unchanged.

The second loop iterates through every agent from W_a and thus, the time complexity for this loop is $\Theta(a)$. Since the second loop occurs inside the first loop, the time complexity so far is $O(g + (g \times a))$.

The third loop iterates through every role from W_r, which has a worst case cardinality of r. Thus, the loop takes $O(r)$. Since the third loops occurs inside the second loop, the time complexity so far is $O(g + (g \times (a \times r)))$.

The `goodness()` function takes $O(c)$ because every capability required by the role has to be checked, which is c in the worst case as all the capabilities in the organization could be required by a role. Since the `goodness()` function is called inside the third loop, the time complexity so far is $O(g + (g \times (a \times (r \times c))))$.

Since the remaining pseudo code is constant time, the time complexity of the greedy algorithm is $O(g + g \times a \times r \times c)$ and can be simplified to $O(g \times a \times r \times c)$. □

D.4 Attributes-Greedy Algorithm

Figure D.4 shows the pseudo code for the attributes-greedy algorithm. The attributes-greedy algorithm starts by computing the set of unassigned goals (g') from the given goals (W_g). Next, the algorithm iterates through each goal (g_i) and obtains the roles (W_r) that can achieve the g_i. It then iterates through each agent (a_i) to determine the best role for

the a_i and g_i, which requires iterating through each role (r_i) and selecting the role with the best `goodness` score. If a_i is capable (i.e., $\beta \neq \emptyset$), the score is compared to the score of the previously saved result (γ). If γ is $\emptyset$ or the score is greater than the score of γ, this agent (a_i) becomes the one that will be assigned to the goal (g_i). The algorithm moves to the next agent and repeats the process until all agents have been checked. The agent ($\gamma.a$) with the highest score is assigned to the goal (g_i). The algorithm moves to the next goal and repeats the process until all goals are processed.

```
function reorganize(W_g, W_a)
  g' ← unassigned(W_g), δ ← ∅
  for each g_i from g' do
    W_r ← achieves(g_i), γ ← ∅
    for each a_i from W_a do
      β ← ∅
      for each r_i from W_r do
        α ← goodness(r_i, a_i, g_i)
        if α > 0 and (β = ∅ or α > β.α) then
          β ← ⟨r_i, a_i, g_i, α⟩
        end if
      end for
      if β ≠ ∅ and (γ = ∅ or β.α > γ.α) then
        γ ← ⟨β.r, β.a, β.g, β.α⟩
      end if
    end for
    δ ← δ ∪ ⟨γ.r, γ.a, γ.g⟩
  end for
  return δ
```

Figure D.4: Pseudo Code – Attributes-Greedy Algorithm

Proof. The time complexity of the attributes-greedy algorithm is $O(g \times a \times r \times (c + n))$.

Let g be the number of goals for the input W_g, a be the number of agents for the input W_a, r be the number of roles in the organization, c be the number of capabilities in the organization, and n be the number of attributes in the organization.

Computing the set of unassigned goals requires checking each goal to determine if it is

assigned. Thus, the `unassigned()` function takes $\Theta(g)$ time.

The first loop iterates through all unassigned goals. In the worst case, all goals from W_g are not assigned and the loop takes $O(g)$ time. The time complexity so far is $O(g + g)$.

The `achieves()` function, which returns a set of roles (W_r), takes constant time, although the cardinality of W_r varies. In the best case, every goal is achieved by 1 role; however, in the worst case, every goal can be achieved by r roles. The time complexity remains unchanged.

The second loop iterates through every agent from W_a and thus, the time complexity for this loop is $\Theta(a)$. Since the second loop occurs inside the first loop, the time complexity so far is $O(g + (g \times a))$.

The third loop iterates through every role from W_r, which has a worst case cardinality of r. Thus, the loop takes $O(r)$. Since the third loops occurs inside the second loop, the time complexity so far is $O(g + (g \times (a \times r)))$.

The `goodness()` function takes $O(c+n)$ because every capability and attribute required by the role has to be checked, which is $c + n$ in the worst case as all the capabilities and attributes in the organization could be required by a role. Since the `goodness()` function is called inside the third loop, the time complexity so far is $O(g + (g \times (a \times (r \times (c+n)))))$.

Since the remaining pseudo code is constant time, the time complexity of the attributes-greedy algorithm is $O(g+g\times a\times r\times(c+n))$ and can be simplified to $O(g\times a\times r\times(c+n))$. □

D.5 Attributes-Enhanced Algorithm

Figure D.5 shows the pseudo code for the attributes-enhanced algorithm. The attributes-enhanced algorithm starts by computing the set of unassigned goals (g') from the given goals (W_g). Next, the algorithm iterates through each goal (g_i) and obtains the roles (W_r) that can achieve g_i. It then iterates through each agent (a_i) to determine the best role for

the a_i and g_i, which requires iterating through each role (r_i) and selecting the role with the best `goodness` score. If a_i is capable (i.e., $\beta \neq \emptyset$) and if the `goodness` score is 1.0, the contribution score is equal to the `goodness` score. However, if the `goodness` score is less than 1.0, the contribution score is computed as shown in Equation 4.20. Then the contribution score is compared to the contribution score of the previously saved result (γ). If γ is $\emptyset$ or the contribution score is greater than one from γ, this agent (a_i) becomes the one that will be assigned to the goal (g_i). The algorithm moves to the next agent and repeats the process until all agents have been checked. The agent ($\gamma.a$) with the highest contribution score is assigned to the goal (g_i). The algorithm moves to the next goal and repeats the process until all goals are processed.

Proof. The time complexity of the attributes-enhanced algorithm is $O(g \times a \times r \times (c+n))$.

Let g be the number of goals for the input W_g, a be the number of agents for the input W_a, r be the number of roles in the organization, c be the number of capabilities in the organization, and n be the number of attributes in the organization.

Computing the set of unassigned goals requires checking each goal to determine if it is assigned. Thus, the `unassigned()` function takes $\Theta(g)$ time.

The first loop iterates through all unassigned goals. In the worst case, all goals from W_g are not assigned and the loop takes $O(g)$ time. The time complexity so far is $O(g + g)$.

The `achieves()` function, which returns a set of roles (W_r), takes constant time, although the cardinality of W_r varies. In the best case, every goal is achieved by 1 role; however, in the worst case, every goal can be achieved by r roles. The time complexity remains unchanged.

The second loop iterates through every agent from W_a and thus, the time complexity for this loop is $\Theta(a)$. Since the second loop occurs inside the first loop, the time complexity so far is $O(g + (g \times a))$.

The third loop iterates through every role from W_r, which has a worst case cardinality

```
function reorganize(W_g, W_a)
  g' ← unassigned(W_g), λ ← ∅
  for each g_i from g' do
    W_r ← achieves(g_i), γ ← ∅
    for each a_i from W_a do
      β ← ∅
      for each r_i from W_r do
        α ← goodness(r_i, a_i, g_i)
        if α > 0 and (β = ∅ or α > β.α) then
          β ← ⟨r_i, a_i, g_i, α⟩
        end if
      end for
      if β ≠ ∅ then
        δ ← β.α
        if δ < 1 then
          δ ← δ × (assignments(β.a) + 1) − previous(β.a)
        end if
        if δ > γ.α then
          γ ← ⟨β.r, β.a, β.g, δ⟩
        end if
      end if
    end for
    λ ← λ ∪ ⟨γ.r, γ.a, γ.g⟩
  end for
  return λ
```

Figure D.5: Pseudo Code – Attributes-Enhanced Algorithm

of r. Thus, the loop takes $O(r)$. Since the third loops occurs inside the second loop, the time complexity so far is $O(g + (g \times (a \times r)))$.

The `goodness()` function takes $O(c+n)$ because every capability and attribute required by the role has to be checked, which is $c + n$ in the worst case as all the capabilities and attributes in the organization could be required by a role. Since the `goodness()` function is called inside the third loop, the time complexity so far is $O(g + (g \times (a \times (r \times (c + n)))))$.

Since the remaining pseudo code is constant time, the time complexity of the attributes-enhanced algorithm is $O(g + g \times a \times r \times (c + n))$ and can be simplified to $O(g \times a \times r \times$

$(c+n))$[1]. □

D.6 Brute Force Algorithm

Since the two brute force algorithms are very similar (except for the calculations of assignment scores and the overall score), Figure D.6 shows the pseudo code for both algorithms. First, the algorithms computes the set of unassigned goals (g') from the given set of goals (W_g). Next, using the unassigned goals (g') and the set of agents (W_a), the two algorithms compute all combinations of assignment sets using their respective assignment score functions; the assignment score function for the OMACS algorithm is defined by Equation 4.28 and the assignment score functions for the CzM algorithm are defined by Equation 4.30 and Equation 4.31 (depending on the task). The two algorithms go through each combination (ϕ_i), which is an assignment set, compute the overall score for the combination using their respective overall score functions and keep track of the best combination. The overall score function for the OMACS algorithm is defined by Equation 4.29 and the overall score function for the CzM algorithm is defined by Equation 4.32.

```
function reorganize(W_g, W_a)
  g' ← unassigned(W_g), β ← ∅
  φ ← combinations(g', W_a)
  for each φ_i from φ do
    α ← score(φ_i)
    if β = ∅ or α > β.α then
      β ← ⟨α, φ_i⟩
    end if
  end for
  return β.φ
```

Figure D.6: Pseudo Code – Brute Force Algorithm

[1]The implementation of Equation 4.20 is constant time. However, the same results can be achieved in a generalized way but the time complexity for the generalized way is unknown.

Proof. The time complexity of the brute force algorithm for the OMACS version is $O(a^g)$ and for the CzM version is $O(a^g)$.

Let g be the number of goals for the input W_g, a be the number of agents for the input W_a, c be the number of capabilities in the organization, and n be the number of attributes in the organization.

Computing the set of unassigned goals requires checking each goal to determine if it is assigned. Thus, the `unassigned()` function takes $\Theta(g)$ time.

The next function (`combinations()`) generates all combinations based on g' and W_a. In the best case there is only one unassigned goal and one agent, so there is only one combination. In the worst case, there are g unassigned goals, a agents, and each unassigned goal can be assigned to a agents. Thus, there are a^g combinations to generate. In order to determine whether an assignment is valid or not, the assignment score functions of the respective algorithms are used. The assignment score functions are linear in terms of c and n because in the worst case all tasks require all capabilities and need all attributes. The assignment score function for the OMACS version takes $O(c)$ time, while the assignment score function for the CzM version takes $O(c+n)$ time. Thus, the `combinations()` function takes $O(a^g \times (c+n))$, where $n = 0$ for the OMACS version. The time complexity so far is $O(g + (a^g \times (c+n)))$, where $n = 0$ for the OMACS version.

The loop iterates through all combinations, which can be as many as a^g. The `score()` function is linear in terms of g because $|\phi_i|$ is at most g. The score for an assignment comes from the assignment score function, which is $O(c+n)$, where $n = 0$ for the OMACS version. Thus, the `score()` function takes $O(g \times (c+n))$, where $n = 0$ for the OMACS version. The remaining pseudo code is constant time. Thus, the loop takes $O(a^g \times (g \times (c+n)))$, where $n = 0$ for the OMACS version. The time complexity so far is $O(g + (a^g \times (c+n)) + (a^g \times (g \times (c+n))))$, where $n = 0$ for the OMACS version.

The time complexity of the brute force algorithm is $O(g+(a^g\times(c+n))+(a^g\times(g\times(c+n))))$

and can be simplified to $O(a^g \times (g \times (c + n)))$, where $n = 0$ for the OMACS version. □

Appendix E

Goal Modification Logs

IAL

```
1 2011/05/27 14:17:43 >> MCC | Incoming modification (state=([1, 1,
   1],false)) for goal (AlphaForm[1])
2 2011/05/27 14:17:43 >> MCC | Goals ([AlphaForm[1] (state=([1, 1,
   1],false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
   0.0), (31.1299, 75.1946, 0.0)], level=*MEDIUM, formation=*WEDGE)])
    affected by modification
3 2011/05/27 14:17:43 >> MCC | Goal (AlphaForm[1] (state=([1, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], level=*MEDIUM, formation=*WEDGE)) is
   assigned to agent (blue: (30,88))
4 2011/05/27 14:17:43 >> MCC | Incoming modification (state=([1, 1,
   1],false)) for goal (BetaForm[1])
5 2011/05/27 14:17:43 >> MCC | Goals ([BetaForm[1] (state=([1, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], level=*MEDIUM, formation=*WEDGE)])
   affected by modification
```

```
6 2011/05/27 14:17:43 >> MCC | Goal (BetaForm[1] (state=([1, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], level=*MEDIUM, formation=*WEDGE)) is
   assigned to agent (green: (32,88))
7 2011/05/27 14:17:43 >> MCC | Incoming modification (state=([1, 1,
   1],false)) for goal (GammaForm[1])
8 2011/05/27 14:17:43 >> MCC | Goals ([GammaForm[1] (state=([1, 1,
   1],false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
   0.0), (31.1299, 75.1946, 0.0)], level=*MEDIUM, formation=*WEDGE)])
    affected by modification
9 2011/05/27 14:17:43 >> MCC | Goal (GammaForm[1] (state=([1, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], level=*MEDIUM, formation=*WEDGE)) is
   assigned to agent (black: (30,89))
10 2011/05/27 14:17:44 >> MCC | Incoming modification (state=([1, 1,
   1],false)) for goal (LeadForm[1])
11 2011/05/27 14:17:44 >> MCC | Goals ([LeadForm[1] (state=([1, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], level=*MEDIUM, formation=*WEDGE)])
   affected by modification
12 2011/05/27 14:17:44 >> MCC | Goal (LeadForm[1] (state=([1, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], level=*MEDIUM, formation=*WEDGE)) is
   assigned to agent (red: (31,86))
13 2011/05/27 14:17:53 >> MCC | Incoming modification (state=([1, 1,
   1],false)) for goal (AlphaMove[1])
```

```
14 2011/05/27 14:17:53 >> MCC | Goals ([AlphaMove[1] (state=([1, 1,
   1],false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
   0.0), (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x
   =31, y=75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)])
   affected by modification
15 2011/05/27 14:17:53 >> MCC | Goal (AlphaMove[1] (state=([1, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
   =75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)) is
   assigned to agent (blue: (30,88))
16 2011/05/27 14:17:53 >> MCC | Incoming modification (state=([1, 1,
   1],false)) for goal (BetaMove[1])
17 2011/05/27 14:17:53 >> MCC | Goals ([BetaMove[1] (state=([1, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
   =75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)])
   affected by modification
18 2011/05/27 14:17:53 >> MCC | Goal (BetaMove[1] (state=([1, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
   =75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)) is
   assigned to agent (green: (32,88))
19 2011/05/27 14:17:53 >> MCC | Incoming modification (state=([1, 1,
   1],false)) for goal (LeadMove[1])
20 2011/05/27 14:17:53 >> MCC | Goals ([LeadMove[1] (state=([1, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
```

```
   =75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)])
   affected by modification
21 2011/05/27 14:17:53 >> MCC | Goal (LeadMove[1] (state=([1, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
   =75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)) is
   assigned to agent (red: (31,86))
22 2011/05/27 14:17:57 >> MCC | Incoming modification (state=([1, 1,
   1],false)) for goal (GammaMove[1])
23 2011/05/27 14:17:57 >> MCC | Goals ([GammaMove[1] (state=([1, 1,
   1],false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
   0.0), (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x
   =31, y=75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)])
   affected by modification
24 2011/05/27 14:17:57 >> MCC | Goal (GammaMove[1] (state=([1, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
   =75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)) is
   assigned to agent (black: (30,89))
25 2011/05/27 14:18:22 >> MCC | Incoming modification (state=([2, 1,
   1],false)) for goal (GammaMove[1])
26 2011/05/27 14:18:22 >> MCC | Goals ([GammaMove[1] (state=([2, 1,
   1],false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
   0.0), (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x
   =31, y=75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)])
   affected by modification
```

```
27 2011/05/27 14:18:22 >> MCC | Goal (GammaMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
   =75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)) is
   assigned to agent (black: (30,89))
28 2011/05/27 14:18:22 >> MCC | Incoming modification (state=([2, 1,
   1],false)) for goal (LeadMove[1])
29 2011/05/27 14:18:22 >> MCC | Goals ([LeadMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
   =75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)])
   affected by modification
30 2011/05/27 14:18:22 >> MCC | Goal (LeadMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
   =75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)) is
   assigned to agent (red: (31,86))
31 2011/05/27 14:18:22 >> MCC | Incoming modification (state=([2, 1,
   1],false)) for goal (BetaMove[1])
32 2011/05/27 14:18:22 >> MCC | Goals ([BetaMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
   =75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)])
   affected by modification
33 2011/05/27 14:18:22 >> MCC | Goal (BetaMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
```

```
    =75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)) is
    assigned to agent (green: (32,88))
34 2011/05/27 14:18:22 >> MCC | Incoming modification (state=([2, 1,
    1],false)) for goal (AlphaMove[1])
35 2011/05/27 14:18:22 >> MCC | Goals ([AlphaMove[1] (state=([2, 1,
    1],false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
    0.0), (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x
    =31, y=75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)])
    affected by modification
36 2011/05/27 14:18:22 >> MCC | Goal (AlphaMove[1] (state=([2, 1, 1],
    false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
     (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
    =75, width=0, height=0], level=*MEDIUM, formation=*WEDGE)) is
    assigned to agent (blue: (30,88))
37 2011/05/27 14:18:24 >> MCC | Incoming modification (status=null,
    points=[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], Task ID=1, location=recon.data.ReconArea
     [x=31, y=75, width=0, height=0], level=LOW, formation=WEDGE) for
    goal (Go To[1])
38 2011/05/27 14:18:24 >> MCC | Goals ([BetaMove[1] (state=([2, 1, 1],
    false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
     (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
    =75, width=0, height=0], level=*LOW, formation=*WEDGE), LeadMove
    [1] (state=([2, 1, 1],false), points=*[(31.2263, 84.1653, 0.0),
    (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)], location=*recon
    .data.ReconArea [x=31, y=75, width=0, height=0], level=*LOW,
    formation=*WEDGE), AlphaMove[1] (state=([2, 1, 1],false), points
```

```
   =*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299,
   75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y=75, width
   =0, height=0], level=*LOW, formation=*WEDGE), GammaMove[1] (state
   =([2, 1, 1],false), points=*[(31.2263, 84.1653, 0.0), (31.2263,
   79.8246, 0.0), (31.1299, 75.1946, 0.0)], location=*recon.data.
   ReconArea [x=31, y=75, width=0, height=0], level=*LOW, formation=*
   WEDGE)]) affected by modification
39 2011/05/27 14:18:24 >> MCC | Goal (BetaMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
   =75, width=0, height=0], level=*LOW, formation=*WEDGE)) is
   assigned to agent (green: (32,88))
40 2011/05/27 14:18:24 >> MCC | Goal (LeadMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
   =75, width=0, height=0], level=*LOW, formation=*WEDGE)) is
   assigned to agent (red: (31,86))
41 2011/05/27 14:18:24 >> MCC | Goal (AlphaMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
   =75, width=0, height=0], level=*LOW, formation=*WEDGE)) is
   assigned to agent (blue: (30,88))
42 2011/05/27 14:18:24 >> MCC | Goal (GammaMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0)], location=*recon.data.ReconArea [x=31, y
   =75, width=0, height=0], level=*LOW, formation=*WEDGE)) is
   assigned to agent (black: (30,89))
```

```
43 2011/05/27 14:18:30 >> MCC | Incoming modification (status=null,
   points=[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
   (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], Task ID=1,
   location=recon.data.ReconArea [x=31, y=69, width=0, height=0],
   level=LOW, formation=WEDGE) for goal (Go To[1])
44 2011/05/27 14:18:30 >> MCC | Goals ([BetaMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE), LeadMove[1] (state=([2, 1, 1],false), points
   =*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299,
   75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*recon.data.
   ReconArea [x=31, y=69, width=0, height=0], level=*LOW, formation=*
   WEDGE), AlphaMove[1] (state=([2, 1, 1],false), points=*[(31.2263,
   84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0),
   (31.0334, 68.9247, 0.0)], location=*recon.data.ReconArea [x=31, y
   =69, width=0, height=0], level=*LOW, formation=*WEDGE), GammaMove
   [1] (state=([2, 1, 1],false), points=*[(31.2263, 84.1653, 0.0),
   (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334,
   68.9247, 0.0)], location=*recon.data.ReconArea [x=31, y=69, width
   =0, height=0], level=*LOW, formation=*WEDGE)]) affected by
   modification
45 2011/05/27 14:18:30 >> MCC | Goal (BetaMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (green: (32,88))
```

```
46 2011/05/27 14:18:30 >> MCC | Goal (LeadMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (red: (31,86))
47 2011/05/27 14:18:30 >> MCC | Goal (AlphaMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (blue: (30,88))
48 2011/05/27 14:18:30 >> MCC | Goal (GammaMove[1] (state=([2, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (black: (30,89))
49 2011/05/27 14:18:35 >> MCC | Incoming modification (state=([3, 1,
   1],false)) for goal (GammaMove[1])
50 2011/05/27 14:18:35 >> MCC | Goals ([GammaMove[1] (state=([3, 1,
   1],false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
   0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location
   =*recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW
   , formation=*WEDGE)]) affected by modification
51 2011/05/27 14:18:35 >> MCC | Goal (GammaMove[1] (state=([3, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (black: (30,89))
```

```
52 2011/05/27 14:18:35 >> MCC | Incoming modification (state=([3, 1,
   1],false)) for goal (LeadMove[1])
53 2011/05/27 14:18:35 >> MCC | Goals ([LeadMove[1] (state=([3, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)]) affected by modification
54 2011/05/27 14:18:35 >> MCC | Goal (LeadMove[1] (state=([3, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (red: (31,86))
55 2011/05/27 14:18:35 >> MCC | Incoming modification (state=([3, 1,
   1],false)) for goal (BetaMove[1])
56 2011/05/27 14:18:35 >> MCC | Goals ([BetaMove[1] (state=([3, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)]) affected by modification
57 2011/05/27 14:18:35 >> MCC | Goal (BetaMove[1] (state=([3, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (green: (32,88))
58 2011/05/27 14:18:35 >> MCC | Incoming modification (state=([3, 1,
   1],false)) for goal (AlphaMove[1])
```

```
59 2011/05/27 14:18:35 >> MCC | Goals ([AlphaMove[1] (state=([3, 1,
   1],false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
   0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location
   =*recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW
   , formation=*WEDGE)]) affected by modification
60 2011/05/27 14:18:35 >> MCC | Goal (AlphaMove[1] (state=([3, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (blue: (30,88))
61 2011/05/27 14:18:55 >> MCC | Incoming modification (state=([4, 1,
   1],false)) for goal (GammaMove[1])
62 2011/05/27 14:18:55 >> MCC | Goals ([GammaMove[1] (state=([4, 1,
   1],false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
   0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location
   =*recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW
   , formation=*WEDGE)]) affected by modification
63 2011/05/27 14:18:55 >> MCC | Goal (GammaMove[1] (state=([4, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (black: (30,89))
64 2011/05/27 14:18:55 >> MCC | Incoming modification (state=([4, 1,
   1],false)) for goal (LeadMove[1])
65 2011/05/27 14:18:55 >> MCC | Goals ([LeadMove[1] (state=([4, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
```

```
    recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
    formation=*WEDGE)]) affected by modification
66 2011/05/27 14:18:55 >> MCC | Goal (LeadMove[1] (state=([4, 1, 1],
    false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
     (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
    recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
    formation=*WEDGE)) is assigned to agent (red: (31,86))
67 2011/05/27 14:18:55 >> MCC | Incoming modification (state=([4, 1,
    1],false)) for goal (BetaMove[1])
68 2011/05/27 14:18:55 >> MCC | Goals ([BetaMove[1] (state=([4, 1, 1],
    false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
     (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
    recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
    formation=*WEDGE)]) affected by modification
69 2011/05/27 14:18:55 >> MCC | Goal (BetaMove[1] (state=([4, 1, 1],
    false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
     (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
    recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
    formation=*WEDGE)) is assigned to agent (green: (32,88))
70 2011/05/27 14:18:55 >> MCC | Incoming modification (state=([4, 1,
    1],false)) for goal (AlphaMove[1])
71 2011/05/27 14:18:55 >> MCC | Goals ([AlphaMove[1] (state=([4, 1,
    1],false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
    0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location
    =*recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW
    , formation=*WEDGE)]) affected by modification
```

```
72 2011/05/27 14:18:55 >> MCC | Goal (AlphaMove[1] (state=([4, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (blue: (30,88))
73 2011/05/27 14:18:58 >> MCC | Incoming modification (state=([5, 1,
   1],false)) for goal (BetaMove[1])
74 2011/05/27 14:18:58 >> MCC | Goals ([BetaMove[1] (state=([5, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)]) affected by modification
75 2011/05/27 14:18:58 >> MCC | Goal (BetaMove[1] (state=([5, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (green: (32,88))
76 2011/05/27 14:18:58 >> MCC | Incoming modification (state=([5, 1,
   1],false)) for goal (LeadMove[1])
77 2011/05/27 14:18:58 >> MCC | Goals ([LeadMove[1] (state=([5, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)]) affected by modification
78 2011/05/27 14:18:58 >> MCC | Goal (LeadMove[1] (state=([5, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
```

```
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (red: (31,86))
79 2011/05/27 14:18:58 >> MCC | Incoming modification (state=([5, 1,
   1],false)) for goal (AlphaMove[1])
80 2011/05/27 14:18:58 >> MCC | Goals ([AlphaMove[1] (state=([5, 1,
   1],false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
   0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location
   =*recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW
   , formation=*WEDGE)]) affected by modification
81 2011/05/27 14:18:58 >> MCC | Goal (AlphaMove[1] (state=([5, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (blue: (30,88))
82 2011/05/27 14:18:58 >> MCC | Incoming modification (state=([5, 1,
   1],false)) for goal (GammaMove[1])
83 2011/05/27 14:18:58 >> MCC | Goals ([GammaMove[1] (state=([5, 1,
   1],false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
   0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location
   =*recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW
   , formation=*WEDGE)]) affected by modification
84 2011/05/27 14:18:58 >> MCC | Goal (GammaMove[1] (state=([5, 1, 1],
   false), points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246, 0.0),
    (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)], location=*
   recon.data.ReconArea [x=31, y=69, width=0, height=0], level=*LOW,
   formation=*WEDGE)) is assigned to agent (black: (30,89))
```

Leader Robot

```
1 2011/05/27 14:17:45 >> red|RCC|LeadForm[1] (formation=*WEDGE, level
  =*MEDIUM, state=([1, 1, 1],false), points=*[(31.2263, 84.1653,
  0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)])
2 2011/05/27 14:17:54 >> red|RCC|LeadMove[1] (formation=*WEDGE, level
  =*MEDIUM, state=([1, 1, 1],false), location=*recon.data.ReconArea
  [x=31, y=75, width=0, height=0], points=*[(31.2263, 84.1653, 0.0),
   (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)])
3 2011/05/27 14:18:20 >> red|Plan|Leader|1|(31.2263, 79.8246, 0.0)
4 2011/05/27 14:18:23 >> red|RCC|LeadMove[1] (formation=*WEDGE, level
  =*MEDIUM, state=([2, 1, 1],false), location=*recon.data.ReconArea
  [x=31, y=75, width=0, height=0], points=*[(31.2263, 84.1653, 0.0),
   (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)])
5 2011/05/27 14:18:25 >> red|RCC|LeadMove[1] (formation=*WEDGE, level
  =*LOW, state=([2, 1, 1],false), location=*recon.data.ReconArea [x
  =31, y=75, width=0, height=0], points=*[(31.2263, 84.1653, 0.0),
  (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)])
6 2011/05/27 14:18:31 >> red|RCC|LeadMove[1] (formation=*WEDGE, level
  =*LOW, state=([2, 1, 1],false), location=*recon.data.ReconArea [x
  =31, y=69, width=0, height=0], points=*[(31.2263, 84.1653, 0.0),
  (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334,
  68.9247, 0.0)])
7 2011/05/27 14:18:33 >> red|Plan|Leader|2|(31.1299, 75.1946, 0.0)
8 2011/05/27 14:18:36 >> red|RCC|LeadMove[1] (formation=*WEDGE, level
  =*LOW, state=([3, 1, 1],false), location=*recon.data.ReconArea [x
  =31, y=69, width=0, height=0], points=*[(31.2263, 84.1653, 0.0),
  (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334,
  68.9247, 0.0)])
```

```
9 2011/05/27 14:18:53 >> red|Plan|Leader|3|(31.0334, 68.9247, 0.0)
10 2011/05/27 14:18:56 >> red|RCC|LeadMove[1] (formation=*WEDGE, level
   =*LOW, state=([4, 1, 1],false), location=*recon.data.ReconArea [x
   =31, y=69, width=0, height=0], points=*[(31.2263, 84.1653, 0.0),
   (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334,
   68.9247, 0.0)])
11 2011/05/27 14:18:59 >> red|RCC|LeadMove[1] (formation=*WEDGE, level
   =*LOW, state=([5, 1, 1],false), location=*recon.data.ReconArea [x
   =31, y=69, width=0, height=0], points=*[(31.2263, 84.1653, 0.0),
   (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334,
   68.9247, 0.0)])
```

Alpha Robot

```
1 2011/05/27 14:17:43 >> blue|RCC|AlphaForm[1] (state=([1, 1, 1],
  false), level=*MEDIUM, points=*[(31.2263, 84.1653, 0.0), (31.2263,
   79.8246, 0.0), (31.1299, 75.1946, 0.0)], formation=*WEDGE)
2 2011/05/27 14:17:53 >> blue|RCC|AlphaMove[1] (state=([1, 1, 1],
  false), level=*MEDIUM, location=*recon.data.ReconArea [x=31, y=75,
   width=0, height=0], points=*[(31.2263, 84.1653, 0.0), (31.2263,
  79.8246, 0.0), (31.1299, 75.1946, 0.0)], formation=*WEDGE)
3 2011/05/27 14:18:22 >> blue|RCC|AlphaMove[1] (state=([2, 1, 1],
  false), level=*MEDIUM, location=*recon.data.ReconArea [x=31, y=75,
   width=0, height=0], points=*[(31.2263, 84.1653, 0.0), (31.2263,
  79.8246, 0.0), (31.1299, 75.1946, 0.0)], formation=*WEDGE)
4 2011/05/27 14:18:24 >> blue|RCC|AlphaMove[1] (state=([2, 1, 1],
  false), level=*LOW, location=*recon.data.ReconArea [x=31, y=75,
  width=0, height=0], points=*[(31.2263, 84.1653, 0.0), (31.2263,
  79.8246, 0.0), (31.1299, 75.1946, 0.0)], formation=*WEDGE)
```

```
5 2011/05/27 14:18:24 >> blue|Plan|Alpha|MEDIUM|1100|LOW|1000
6 2011/05/27 14:18:30 >> blue|RCC|AlphaMove[1] (state=([2, 1, 1],
  false), level=*LOW, location=*recon.data.ReconArea [x=31, y=69,
  width=0, height=0], points=*[(31.2263, 84.1653, 0.0), (31.2263,
  79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
  formation=*WEDGE)
7 2011/05/27 14:18:35 >> blue|RCC|AlphaMove[1] (state=([3, 1, 1],
  false), level=*LOW, location=*recon.data.ReconArea [x=31, y=69,
  width=0, height=0], points=*[(31.2263, 84.1653, 0.0), (31.2263,
  79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
  formation=*WEDGE)
8 2011/05/27 14:18:55 >> blue|RCC|AlphaMove[1] (state=([4, 1, 1],
  false), level=*LOW, location=*recon.data.ReconArea [x=31, y=69,
  width=0, height=0], points=*[(31.2263, 84.1653, 0.0), (31.2263,
  79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
  formation=*WEDGE)
9 2011/05/27 14:18:58 >> blue|RCC|AlphaMove[1] (state=([5, 1, 1],
  false), level=*LOW, location=*recon.data.ReconArea [x=31, y=69,
  width=0, height=0], points=*[(31.2263, 84.1653, 0.0), (31.2263,
  79.8246, 0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
  formation=*WEDGE)
```

Beta Robot

```
1 2011/05/27 14:17:44 >> green|RCC|BetaForm[1] (state=([1, 1, 1],
  false), level=*MEDIUM, points=*[(31.2263, 84.1653, 0.0), (31.2263,
   79.8246, 0.0), (31.1299, 75.1946, 0.0)], formation=*WEDGE)
2 2011/05/27 14:17:54 >> green|RCC|BetaMove[1] (location=*recon.data.
  ReconArea [x=31, y=75, width=0, height=0], state=([1, 1, 1],false)
```

```
   , level=*MEDIUM, points=*[(31.2263, 84.1653, 0.0), (31.2263,
   79.8246, 0.0), (31.1299, 75.1946, 0.0)], formation=*WEDGE)
 3 2011/05/27 14:18:23 >> green|RCC|BetaMove[1] (location=*recon.data.
   ReconArea [x=31, y=75, width=0, height=0], state=([2, 1, 1],false)
   , level=*MEDIUM, points=*[(31.2263, 84.1653, 0.0), (31.2263,
   79.8246, 0.0), (31.1299, 75.1946, 0.0)], formation=*WEDGE)
 4 2011/05/27 14:18:25 >> green|RCC|BetaMove[1] (location=*recon.data.
   ReconArea [x=31, y=75, width=0, height=0], state=([2, 1, 1],false)
   , level=*LOW, points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
    0.0), (31.1299, 75.1946, 0.0)], formation=*WEDGE)
 5 2011/05/27 14:18:25 >> green|Plan|Beta|MEDIUM|1100|LOW|1000
 6 2011/05/27 14:18:31 >> green|RCC|BetaMove[1] (location=*recon.data.
   ReconArea [x=31, y=69, width=0, height=0], state=([2, 1, 1],false)
   , level=*LOW, points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
    0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
   formation=*WEDGE)
 7 2011/05/27 14:18:36 >> green|RCC|BetaMove[1] (location=*recon.data.
   ReconArea [x=31, y=69, width=0, height=0], state=([3, 1, 1],false)
   , level=*LOW, points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
    0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
   formation=*WEDGE)
 8 2011/05/27 14:18:56 >> green|RCC|BetaMove[1] (location=*recon.data.
   ReconArea [x=31, y=69, width=0, height=0], state=([4, 1, 1],false)
   , level=*LOW, points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
    0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
   formation=*WEDGE)
```

```
9 2011/05/27 14:18:59 >> green|RCC|BetaMove[1] (location=*recon.data.
  ReconArea [x=31, y=69, width=0, height=0], state=([5, 1, 1],false)
  , level=*LOW, points=*[(31.2263, 84.1653, 0.0), (31.2263, 79.8246,
   0.0), (31.1299, 75.1946, 0.0), (31.0334, 68.9247, 0.0)],
  formation=*WEDGE)
```

Gamma Robot

```
1 2011/05/27 14:17:47 >> black|RCC|GammaForm[1] (points=*[(31.2263,
  84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)],
  state=([1, 1, 1],false), level=*MEDIUM, formation=*WEDGE)
2 2011/05/27 14:18:01 >> black|RCC|GammaMove[1] (points=*[(31.2263,
  84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)],
  location=*recon.data.ReconArea [x=31, y=75, width=0, height=0],
  state=([1, 1, 1],false), level=*MEDIUM, formation=*WEDGE)
3 2011/05/27 14:18:25 >> black|RCC|GammaMove[1] (points=*[(31.2263,
  84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)],
  location=*recon.data.ReconArea [x=31, y=75, width=0, height=0],
  state=([2, 1, 1],false), level=*MEDIUM, formation=*WEDGE)
4 2011/05/27 14:18:28 >> black|RCC|GammaMove[1] (points=*[(31.2263,
  84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0)],
  location=*recon.data.ReconArea [x=31, y=75, width=0, height=0],
  state=([2, 1, 1],false), level=*LOW, formation=*WEDGE)
5 2011/05/27 14:18:28 >> black|Plan|Gamma|MEDIUM|1100|LOW|1000
6 2011/05/27 14:18:33 >> black|RCC|GammaMove[1] (points=*[(31.2263,
  84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0),
  (31.0334, 68.9247, 0.0)], location=*recon.data.ReconArea [x=31, y
  =69, width=0, height=0], state=([2, 1, 1],false), level=*LOW,
  formation=*WEDGE)
```

```
7 2011/05/27 14:18:39 >> black|RCC|GammaMove[1] (points=*[(31.2263,
  84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0),
  (31.0334, 68.9247, 0.0)], location=*recon.data.ReconArea [x=31, y
  =69, width=0, height=0], state=([3, 1, 1],false), level=*LOW,
  formation=*WEDGE)
8 2011/05/27 14:18:59 >> black|RCC|GammaMove[1] (points=*[(31.2263,
  84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0),
  (31.0334, 68.9247, 0.0)], location=*recon.data.ReconArea [x=31, y
  =69, width=0, height=0], state=([4, 1, 1],false), level=*LOW,
  formation=*WEDGE)
9 2011/05/27 14:19:01 >> black|RCC|GammaMove[1] (points=*[(31.2263,
  84.1653, 0.0), (31.2263, 79.8246, 0.0), (31.1299, 75.1946, 0.0),
  (31.0334, 68.9247, 0.0)], location=*recon.data.ReconArea [x=31, y
  =69, width=0, height=0], state=([5, 1, 1],false), level=*LOW,
  formation=*WEDGE)
```