DTAACS: Distributed Task Allocation for Adaptive Computational Systems based on Organization Knowledge

by

Jorge L. Valenzuela

M.S., Kansas State University, 2000
B.S., ITESM, Monterrey, Mexico, 1990

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
2014

Abstract

The Organization-Based Multi-Agent Systems (OMAS) paradigm is an approach to address the challenges posed by complex systems. The complexity of these systems, the changing environment where the systems are deployed, and satisfying higher user expectations are some of current requirements when designing OMAS. For the agents in an OMAS to pursue the achievement of a common goal or task, a certain level of coordination and collaboration occurs among them. An objective in this coordination is to make the decision of *who does what*. Several solutions have been proposed to answer this task allocation question. The majority of the solutions proposed fall in the categories of marked-based approaches, reactive systems, or game theory approaches. A common fact among these solutions is the system information sharing among agents, which is used *only* to keep the participant agent informed about other agents activities and mission status.

To further exploit and take advantage of this system information shared among agents, a framework is proposed to use this information to answer the question *who does what*, and reduce the communication among agents. DTAACS-OK is a distributed knowledge-based framework that addresses the Single Agent Task Allocation Problem (SAT-AP) and the Multiple Agent Task Allocation Problem (MAT-AP) in cooperative OMAS. The allocation of tasks is based on an identical organization knowledge posses by all agents in the organization. DTAACS-OK differs with current solutions in that (a) it is not a marked-based approach where task are auctioned among agents, or (b) it is not based on agents behaviour, where the action or lack of action of an agent cause the reaction of other agents in the organization.

DTAACS: Distributed Task Allocation for Adaptive Computational Systems based on Organization Knowledge

by

Jorge L. Valenzuela

M.S., Kansas State University, 2000
B.S., ITESM, Monterrey, Mexico, 1990

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
2014

Approved by:

Major Professor
Scott A. DeLoach

Copyright

Abstract

The Organization-Based Multi-Agent Systems (OMAS) paradigm is an approach to address the challenges posed by complex systems. The complexity of these systems, the changing environment where the systems are deployed, and satisfying higher user expectations are some of current requirements when designing OMAS. For the agents in an OMAS to pursue the achievement of a common goal or task, a certain level of coordination and collaboration occurs among them. An objective in this coordination is to make the decision of *who does what*. Several solutions have been proposed to answer this task allocation question. The majority of the solutions proposed fall in the categories of marked-based approaches, reactive systems, or game theory approaches. A common fact among these solutions is the system information sharing among agents, which is used *only* to keep the participant agent informed about other agents activities and mission status.

To further exploit and take advantage of this system information shared among agents, a framework is proposed to use this information to answer the question *who does what*, and reduce the communication among agents. DTAACS-OK is a distributed knowledge-based framework that addresses the Single Agent Task Allocation Problem (SAT-AP) and the Multiple Agent Task Allocation Problem (MAT-AP) in cooperative OMAS. The allocation of tasks is based on an identical organization knowledge posses by all agents in the organization. DTAACS-OK differs with current solutions in that (a) it is not a marked-based approach where task are auctioned among agents, or (b) it is not based on agents behaviour, where the action or lack of action of an agent cause the reaction of other agents in the organization.

Table of Contents

Table of Contents vi

List of Figures x

List of Tables xi

Acknowledgements xii

Dedication xiii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Thesis Statement . . . 3
- 1.3 Contributions . . . 4
- 1.4 Overview of Research Approach . . . 4
  - 1.4.1 Problem Description, Abstraction, and Models . . . 4
  - 1.4.2 Distributed Systems . . . 6
  - 1.4.3 Framework Design . . . 7
  - 1.4.4 Evaluation . . . 7
- 1.5 Assumptions . . . 8
- 1.6 Summary . . . 9

2 Problem Formulation 10
- 2.1 Task Allocation Problem . . . 10
  - 2.1.1 Tasks . . . 11
  - 2.1.2 Problem Description . . . 15
  - 2.1.3 Single Agent Task Allocation Problem . . . 15
  - 2.1.4 Multiple Agent Task Allocation Problem . . . 17
- 2.2 Mission, Tasks and Agent Representations . . . 18
- 2.3 Problem Statement . . . 20
- 2.4 Summary . . . 22

3 Background 23
- 3.1 Mathematical Background . . . 23
  - 3.1.1 Sets and Combinations . . . 24
  - 3.1.2 Operations Research . . . 24
    - 3.1.2.1 OR Models . . . 25
  - 3.1.3 Combinatorial Optimization . . . 25

3.1.3.1 The Fundamental Algorithm . . . 26
3.1.4 Linear Programming . . . 27
3.1.4.1 Solution Techniques for Combinatorial Optimization Problems 28
3.1.5 Set Partition and Set Coverage . . . 29
3.2 Computer Science Background . . . 30
3.2.1 Distributed Systems . . . 30
3.2.1.1 Election Algorithms and Mutual Exclusion . . . 31
3.2.1.2 Data Replication . . . 33
3.2.1.3 Concurrency Control . . . 33
3.2.1.4 Replica Management . . . 34
3.3 Agents, MultiAgent Systems, and OMAS . . . 36
3.3.1 Agents . . . 36
3.3.2 Multi-Agent Systems . . . 37
3.3.3 Organization MAS . . . 39
3.4 OMACS . . . 39
3.5 Conclusion . . . 40

4 DTAACS-OK 42
4.1 DTAACS-OK Components . . . 43
4.1.1 A General Overview of Mission Execution . . . 43
4.2 Distributed Transaction Component . . . 45
4.2.1 Distributed Transaction Component . . . 46
4.2.2 Transaction Generator . . . 47
4.2.3 Transaction Manager . . . 48
4.2.3.1 Distributed Knowledge Systems . . . 50
4.2.3.2 Replica Control . . . 50
4.2.3.3 Transaction Atomicity . . . 51
4.3 Distributed Organization Knowledge Component . . . 54
4.3.1 Task Set Selection Module . . . 54
4.3.2 Organization Information Module . . . 56
4.4 Distributed Task Allocation Component . . . 57
4.4.1 Allocation Algorithms . . . 58
4.4.1.1 WorkInMission Algorithm . . . 58
4.4.1.2 AllocateTasks Algorithm . . . 60
4.4.1.3 GetBestAgent Algorithm . . . 62
4.4.2 Utility Function and Assignment Policies . . . 64
4.4.3 Utility Criteria Entities . . . 64
4.5 Agent's Local Information Component . . . 65
4.6 Summary . . . 65

5 Coalitions in DTAACS-OK **67**
5.1 Motivation and Problem Illustration . . . 68
5.1.1 The Site Clearing Problem . . . 68
5.1.2 Tasks Taxonomy . . . 70
5.2 Coalitions in DTAACS-OK . . . 72
5.3 Problem Statement . . . 74
5.4 Coalition Algorithms . . . 75
5.4.1 A General Overview of Candidate Coalitions Generation . . . 75
5.4.2 GetBestCoalition Algorithm . . . 76
5.4.3 MainCoalitionFormation Algorithm . . . 77
5.4.4 CoalitionsForTask Algorithm . . . 78
5.4.5 FilterCandidateAgents Algorithm . . . 79
5.5 Summary . . . 80

6 DTAACS-OK Empirical Evaluations **81**
6.1 DTAACS-OK for HuRT-IED . . . 81
6.1.1 Motivation . . . 81
6.1.2 Mission and Task specification . . . 82
6.1.3 General Scenario Description . . . 83
6.1.3.1 Scenario Evaluation based on the SOs types . . . 86
6.1.4 Particular Scenario Specification . . . 86
6.2 DTAACS-OK for Collaborative Assembling Objects . . . 94
6.2.1 Motivation . . . 94
6.2.2 Mission and Task specification . . . 94
6.2.3 General Scenario Description . . . 97
6.2.4 Particular Scenario Specification . . . 99
6.3 DTAACS-OK versus DEMiRF-CF . . . 106
6.3.1 Motivation . . . 106
6.3.2 General Scenario Description . . . 106
6.3.3 Particular Scenario Specification . . . 107
6.4 Summary . . . 108

7 Related Work **109**
7.1 Market-Based Approaches . . . 111
7.1.1 M+ . . . 111
7.1.2 TraderBots . . . 112
7.1.3 Incremental Multi-Robot Task Selection . . . 113
7.2 Markov Decision Problem . . . 113
7.2.1 Decentralized Dynamic Task Allocation . . . 114
7.2.2 Modeling Task Allocation Using a Decision Theoretical Model . . . 115
7.3 Other Approaches . . . 116
7.3.1 Alliance . . . 116

7.3.2 Distributed Task Allocation in MAS based on Decision Support Module 117
7.4 Coalition Formation and Task Allocation . . . 119
7.4.1 Task Allocation via Coalition Formation . . . 119
7.4.2 Multi-Robot Coalition Formation . . . 119
7.4.3 Bayesian Model-Based Coalition Formation Approach . . . 120
7.4.4 Building Coalitions Through Automated Task Solution Synthesis . . . 120

8 Discussion And Conclusion **122**
8.1 Prevailing and Relevant Solutions . . . 122
8.2 DTAACS-OK: The Framework . . . 124
8.2.1 DTAACS-OK solution to the SAT-AP and MAT-AP . . . 124
8.3 Future Work . . . 128
8.3.1 Identical Organization Knowledge . . . 128
8.3.2 Coalitions . . . 128
8.4 Conclusion . . . 129
8.5 Summary . . . 130

Bibliography **136**

List of Figures

2.1 Tight Coordination-Simple Task . . . 13
2.2 Tight Coordination-Complex Tasks . . . 14
2.3 HuRT IED Mission . . . 19

3.1 OMACS Metamodel . . . 41

4.1 DTAACS-OK Components . . . 44
4.2 Distributed Transaction Component Diagram . . . 47
4.3 Transaction Generator . . . 48
4.4 Transaction Example . . . 48
4.5 Distributed Organization Knowledge Component . . . 56
4.6 Distributed Task Allocation Component . . . 58

5.1 Site Clearing Task Diagram . . . 71

6.1 HuRT-IED Scenario . . . 84
6.2 HuRT-IED Tree Mission Representation . . . 85
6.3 HuRT-IED Average Message Sent all SOs are G . . . 88
6.4 HuRT-IED Average Message Sent all SOs are IEDs identifiable only by Human Agent . . . 89
6.5 HuRT-IED Average Message Sent all SOs are Mix . . . 90
6.6 HuRT-IED Average Message Sent with Task Reallocation all SOs are G . . . 92
6.7 HuRT-IED Average Message Sent with Task Reallocation all SOs are IEDs identifiable only by Human Agent . . . 92
6.8 HuRT-IED Average Message Sent with Task Reallocation 15% all SOs are IEDs identifiable only by Human Agent . . . 93
6.9 CAO Scenario . . . 95
6.10 CAO Tree Mission Representation . . . 98
6.11 CAO Average Messages Sent for 0%, 15%, 30%, and 60% Probability of Message Drop . . . 101
6.12 CAO Average Assignments for 0%, 15%, 30%, and 60% Probability of Task Failure . . . 102
6.13 CAO Average Messages Sent Under Task Failure . . . 103
6.14 CAO Average Assignments Under Task and Communication Failure (15%) . . . 104
6.15 CAO Average Messages Sent Under Task and Communication Failure (30%) 105
6.16 DTAACS-OK versus DEMiRF-CF CAO No Message Drop . . . 107
6.17 DTAACS-OK versus DEMiRF-CF 15% Probability Message Drop . . . 108

List of Tables

4.1 Required Events From Agent Task Execution Component 49
4.2 DTAACS-OK Transaction Types 55
4.3 Attributes representing an Agent in the Organization 57
4.4 Agent Status 57
4.5 Capabilities in the Organization 57
4.6 Agent's Information Examples 66

5.1 Clearing Site Objects 70
5.2 Clearing Site Agents 70
5.3 Group Criteria 71
5.4 Task Taxonomy for Coalition Formation 72

6.1 Agent Types 85
6.2 Independent Variables 87
6.3 Dependent Variables 88
6.4 Object Types 99
6.5 Robot Types 99
6.6 CAO Independent Variables 100
6.7 DTAACS-OK and DEMiRF-CF Similarities 106

Acknowledgments

This has been a long journey, and not an easy one. I want to take this opportunity to thank and acknowledge all the people that, in one way or other, helped and supported me along the way. First, I would like to thank my Major advisor Dr. Scott DeLoach, he has provided the guidance and freedom to pursue my degree, with the patients my nontraditional grad student situation required. I respect and appreciate his patience in reading my dissertation as many time as needed. Thanks a lot Dr. DeLoach. I also want to thank my committee members, Dr. Singh and Dr. Neilsen for their comments, questions, and feedback that helped me during my research. Also, thanks to Dr. Bala Natarajan and Dr. Tim Bolton, for their comments and interest in my research.

Dedication

This dissertation is dedicated to my children, Alejandro and Victor, they fulfill my life and make this a life worth lived. Additionally, I dedicate this work to my family, they always believed in me and never gave up on me, their expectations and desired for me to succeed kept me trying. To my close and dear friends Jaime and Manuel, they were always there for me when the frustrations arrived, and all those difficult moments I went thorough during this journey. Furthermore, this work is dedicated to Tere Ortega, her love for science and knowledge always inspired me, and I will always admire in her.

Chapter 1

Introduction

The rapid and continuous pace of technological advances, particularly for digital devices and their operating software, contribute to increased user expectations in regards to adaptability, autonomy, robustness, and security. This progress in technology development also prompts system designers to develop efficient solutions for challenging problems in application domains where time constraints, communication limitations, remote human interaction, and adverse and dynamic environments are typical characteristics. Systems proposed to satisfy these demands are rather complex and typically executed in a distributed way among diverse computational systems, thus requiring these systems to decide which computational entity should work on a particular task. Consequently, this dissertation focus on the allocation of tasks in distributed heterogenous systems.

1.1 Motivation

In Multi-Agent Systems (MAS), particularly MAS with Organization Theory, techniques are incorporated to address scalability, adaptability and systematic design. In addition, Organization-Based Multi-Agent Systems (OMAS) paradigms are used to design complex systems for deployment in challenging application domains and the simultaneous satisfaction of increasing user expectations. MAS, and particularly OMAS, are specifically designed to allow participant agents to cooperate and/or collaborate to achieve a common goal or task.

For agents in an OMAS to pursue achievement of a common goal, a certain level of

coordination must occur.. An objective of this coordination is to decide *who does what*. In various challenging application domains such as *Search and Rescue*, *Hazardous Material Disposal*, and *Military Applications*, an efficient distributed task allocation solution is critical to system performance and solution outcome.

Search and Rescue. The Disaster Response application area has received special attention over the years. Several solutions have been proposed and tested for communication, time response, and adaptability performance in scenarios with different adversity levels (Yellow, Orange, and Red arenas as defined by the National Institute of Standards and Technology). Adversity is defined as the obstacles a robot may encounter and the level of communication challenges among the robots. Solutions proposed by Botelho and Alami [15], Zlot and Stentz[57], and Sanem and Balch [40] are distributed marked-based approaches. Even though these solutions claim to be distributed, the coordinating agent of the bidding process can be considered a centralized decision point. The communication cost for these solutions is determined by communication that occurs through the bidding process. Besides communication among agents during the task allocation process, the agents also share system information to track progress and status of the mission. This sharing of information among agents is essential in exploring a new approach to the task allocation problem captured in this research. The type of tasks Botelho and Alami's [15] and Sanem and Balch's [40] solutions can handle are simple tasks that may be reassigned, thus increasing communication costs. In Chapter 5, a more detailed discussion of these approaches is offered.

Military Applications. The military domain presents a very challenging environment due to ever-present uncertainty and drastic changes. The agents in a MAS deployed in this domain need to adapt rapidly to sudden and sometimes catastrophic changes in the environment, and overcome any drawbacks a team member may face. Beautement et. al. [5] suggests that any system designed for the military domain needs to handle three key issues :

- *Availability and Reliability.* Systems may run continually for long periods of time. Updates, debug time, and restart activities are generally not possible.

- *Avoid Single Point of Failure.* Solutions need to be distributed and secure while meeting efficiency objectives.

- *Enable Robustness and Resilience from the Start of Execution.* Systems need to be ready for malicious interferences and attacks, as well as possible system failures.

These three characteristics require agents in the MAS to coordinate their actions in such a way that resources such as battery power and communication bandwidth are used efficiently and a secure, robust and distributed solution is provided. In the case of multi-robot systems, two applications are of particular interest: Reconnaissance and Improvised Explosive Device (IED) detection and defusing. In Chapter 6, an IED application is used to test the proposed framework and the results are discussed in detail.

A majority of current solutions for the task allocation problem are market-based solutions where, besides communication cost of the bidding process, there is also a communication cost to share and maintain knowledge of the system state [15, 40, 57]. While studying these market-based approaches I was intrigued to find out whether this shared system information possessed the property of *one copy serializability* [55], would be enough to address the task allocation problem. The question asked was: if agents in an OMAS share identical system state information, are agents able to answer the question *who does what?*.

To answer the question, this research proposes Distributed Task Allocation in Adaptive Computational Systems based on Organization Knowledge(DTAACS-OK) framework to handle the task allocation problem in OMAS.

1.2 Thesis Statement

By maintaining identical organization knowledge in each agent, an OMAS is able to more efficiently allocate tasks and reduce communication costs as compared to market-based ap-

proaches.

1.3 Contributions

The contributions of this research are:

- A distributed task-allocation framework for OMAS that provides reliability and adaptability required by hazardous and dangerous application domains.
- A set of algorithms to form coalitions when a task requires more than one agent to be executed. Communication cost due to coalition formation is reduced when compared to current approaches.
- A set of algorithms for task allocation that reduce necessary communication when compared to current marked-based approaches.

1.4 Overview of Research Approach

The main objective of this research is to offer a non-communication intensive yet efficient distributed solution to the task allocation problem in an OMAS. In this section, an overview of the approach in pursuing this solution is presented.

1.4.1 Problem Description, Abstraction, and Models

The Task Allocation Problem can be formulated in various ways (see Chapter 2) depending on application domain, user requirements, and optimization objective. The first step in this research is to define and specify the scope of the problem. Chapter 2 defines the inputs, systems, tasks, and optimization objectives considered by this work. The next step in the research is to utilize mathematical models to represent the task allocation problem.

In situations where more than one computational entity shares resources, tasks, or valuable artifacts, the question of what entity is allocated a resource or assigned a task can be generalized as *who gets what?*. The answer to this question is not trivial and, in some cases,

the *what* can be decomposed, posing a slightly different question: *who gets what part of the what?*. The answer to these questions significantly impacts performance and outcome of the system, and a careful decision process must be followed. The *task allocation problem* has been addressed in the pass in several areas besides computer science, as well as different areas within computer science. A brief overview of these areas is given below.

Operational Research (OR). In OR, the objective is to improve the process of decision-making. OR is multidisciplinary but relies heavily on mathematical sciences, such as mathematical modeling, statistical analysis, and optimization. Problems that require maximization or minimization of an objective and are restricted by certain conditions are modeled in OR by *Linear Programming*, which specifies structure of the problem and denotes the way problems with such structure are solved. A typical problem analyzed in OR is the *allocation problem*. Given the sets I and J, x_{ij} represents that i is assigned to j and c_{ij} the cost of the assignment. If I and J have same cardinality, the problem can be modeled with the linear programming formulation described below:

$$\begin{aligned} minimize\ M\ = \sum_i \sum_j c_{ij}\, x_{ij} \quad subject\ to: \\ \sum_j x_{ij}\ =\ 1\ for\ all\ i \\ \sum_i x_{ij}\ =\ 1\ for\ all\ j \\ x_{ij}\ =\ 0\ or\ 1\ for\ all\ i\ and\ j \end{aligned} \tag{1.1}$$

This model fits most of the interesting problems tackled by OMAS, thus it is considered in the solution proposed in this Dissertation.

Markov Decision Problem (MDP). Consider a scenario where an agent that is situated in an environment is pursuing a goal. This scenario can be described as an entity with capability to change states of the environment, where the goal is one of those environment states. An MDP consists of a set of states S, (where s_1 is the initial state), a transition function $T(s, a, s')$, and a reward function $r : S \rightarrow R$. If it is assumed the agent is the only

entity that can make changes in the environment, this model fits the scenario described above. To model a MAS as a MDP, the fact that any agent can modify the environment requires that the transition function incorporate possible changes other agents make to the environment.

Distributed Systems. Some commonly shared resources in distributed systems include I/O channels, buffers, data files, and computational power. The entities that comprise a distributed system need to implement a mechanism to share resources and avoid deadlock and starvation. The algorithms proposed by Rhee [38] attemt to minimize the time for a participant to acquire all required resources. Rhee defines a possible model for system P that consists of a finite or infinite set of processes (p_i) and each process as a finite state automaton. Communication among processes is modeled as a special process called *network*. The finite state automaton is specified by a guarded command $(B_i \rightarrow A_i)$ with two parts. The first part (B_i) represents a boolean expression or message reception and the second part (A_i) represents a finite list of action statements that consist of multiple local steps. Each process possesses a buffer that communicates with the rest of the processes by adding/removing messages to its buffer. The network schedules delivery of messages sent among the processes and delivers the messages by pulling a message from a particular buffer and placing it into destination buffers. The system model also includes the definition of a *sequence*, which is the vector $C = \{q_1, q_2, ..\}$; where q_i is the local state of process p_i The system executes a sequence of configurations asynchronously.

The previous paragraphs give a brief overview of ways the task allocation problem has been modeled. In Chapter 2, an extended discussion related to the specific model used in the proposed framework is presented.

1.4.2 Distributed Systems

Once the task allocation problem statement is presented and a mathematical model for the problem is specified, it is important to discuss some areas of distributed systems that will

be considered in the solution.

Information Consistency. The proposed solution is based on identical organization knowledge in each agent (See Section 2.2.1); the methods and techniques used to provide information consistency in a distributed system are key to this research. Several protocols to support replicated data are available. Chapter 2 includes a more detailed discussion of these protocols, describing the *ways* data can be replicated, *strategies* to update the replicas, and how the network partitioning problem is handled by protocols.

Election Algorithms. To keep and maintain identical organization knowledge in each agent, selection of the agent to execute the next update is a key part of the knowledge update mechanism. Several protocols studied in distributed system can provide a solution to this requirement.

1.4.3 Framework Design

The next step in this research approach is to consider an architectural design that can be easily integrated into current and available MAS design methodologies/framewroks. The goal is to keep this architecture as modular and decoupled as possible.

1.4.4 Evaluation

Since the goal of this dissertation is to provide a low communication yet efficient distributed solution to the task allocation problem in OMAS, evaluation of the framework proposed in this work was focused on how it behaves in regards to communication cost and how communication degradation affects mission achievement. The framework was implemented in two applications which are examples of collaborative systems in military and manufacturing domains. In the first application, HuRT-IED, a team of robots and a human agent have the objective of clearing a road intersection from possible IEDs seeded by the enemy. In the Collaborative Object Assembling (COAApp) application, a team of robots has the mission to build an object from parts that must be assemble in a predefined order. Both experiments are controlled experiments, meaning that the framework was evaluated in regards to

communication cost for (1) task allocation, (2) coalition formation, and elapsed time for (3) mission achievement. The tests used (a) error-free communication and (b) stressed communication conditions, where communication capability of each agent in the organization deteriorates, resulting in lost messages.

Implementation of these applications was conducted in a simulator developed at Kansas State University's Multi-Agent and Cooperative Reasoning Laboratory. The Cooperative Robotics Organization Simulator (CROS) [31] supports the execution of OMAS applications in a controlled environment.

1.5 Assumptions

MAS and multi-robot systems have a wide range of applicability. Evaluation of the framework proposed in this work is done in application domains where direct human participation in the mission is of high risk so that remote human interaction with the agents is of great value. The following assumptions help stress aspects of the framework that are of interest in evaluating in this research within a reasonable time frame.

1. **Discrete Tasks**. All tasks have specific *start* and *finish* states. The main reason for this assumption is to help specify an application termination criteria.

2. **Priority Tasks**. All tasks in the mission have a predefined priority. This assumption allows ordering of task execution. If no priority is given, a random selection of available tasks is required.

3. **Heterogeneous Robot Teams**. One way to provide robustness to a multi-robot system is to have the capabilities/resources distributed among different robots. If the proposed framework is used in a homogeneous system, discrimination criteria, such as *work load*, would be required to determine allocation of tasks.

4. **Full communication**. All agents can communicate directly or indirectly with each other. This assumption does not eliminate the possibility that one or more agents fail

to receive messages.

5. **Closed Systems**. The number of agents in a team does not increase. An agent can leave the organization but cannot reenter the organization.

1.6 Summary

In this chapter, the importance of a distributed, low cost communication solution to the task allocation problem in OMAS is discussed. First, a possible solution to this problem is presented. Next, steps followed to provide the solution is presented as a framework that each agent in the organization implements and executes. Finally, a brief description of how the proposed framework was evaluated is discussed.

The reminder of this dissertation is as follows: Because the task allocation problem can be formulated in different ways based on factors such as application domain and optimization objectives, Chapter 2 presents a formal definition of the problem addressed in this research. Chapter 3 includes a discussion of related technologies that support development of the framework proposed in this research. In Chapter 4, different components of the DTAACS-OK framework and algorithms to handle the Single-Agent Task Allocation problem are offered. In Chapter 5, a discussion of how the Multi-Agent Task Allocation problem is handled in the framework proposed in this work is presented. In Chapter 6, a preliminary empirical evaluation of the framework is presented, including descriptions and results of the proposed evaluation. Chapter 7 discusses the most relevant solutions currently proposed to similar problems. Finally, a discussion and a conclusion regarding topics addressed in this Dissertation are presented in Chapter 8.

Chapter 2

Problem Formulation

The mathematical models used to formulate task allocation among individuals are diverse and depend on the area of study. When studying social insect colonies, the models are mostly based on interactions and behavior [29]. In Microeconomics, factors like consumer-supplier, price, budget and demand shape the models for allocating resources to individuals [22]. In Computer Sciences, the different mathematical models to address task allocation refer to concepts from other disciplines, like the ones mentioned above. In this research, the formulation of the task allocation problem incorporates techniques and concepts from Operations Research and Economics. The problem is classified based on the categories defined in [28]. A *Task* is a broad and general abstraction used in MAS and in Section 2.1, the types of tasks considered in this research are specified. Mission and tasks representations are described in Section 2.2, and the actual problem statement is posed in Section 2.3.

2.1 Task Allocation Problem

The task allocation problem in MAS can be categorized based on the number of tasks an agent can perform simultaneously, the number of agents needed to execute a task, and whether the application considers future states of the system when allocating tasks [28].

2.1.1 Tasks

As mentioned above a task is a general abstraction used in MAS. In the area of insect societies, Oster and Wilson define a task as a set of behaviors that must be performed to achieve some purpose of the colony. Wooldridge [54] talks about tasks as a way to tell an agent what to do, but not how to do it. The task definition used in this research combines both, the *what* which is something to be achieved or the desirable state of the system to be reached, and the *how* which is the set of steps to achieve it. Depending on whether or not the set of steps can be divided into subsets such that each subset achieves part of the desirable system state, two task definitions are defined below:

Definition 2.1. Simple Task

A *Simple Task* (ST) is a task which set of steps cannot be grouped into subset such that, each subset achieves part of the desirable state.

Zlot et al. [57] address the allocation problem for complex tasks. The authors define complex tasks as follow:

Definition 2.2. Complex Task

Complex Tasks (CT) are tasks in which a set of steps can be grouped into subsets such that each of the subsets achieve part of the desirable state in the environment.

Complex tasks are usually introduced into the system as a rooted task tree, where the tasks are related by a parent-child relationship, and each child is a refinement of its more abstract parent [15, 40, 57]. The subtasks in [57] are related to their parent by AND and OR relationships and relate to each other by a *precedence* relationship. The AND and OR relationships specifies what tasks need to be completed so the parent is achieved. The precedence relationship restricts the order of task execution.

Tasks can also be classified as tightly or loosely coordinated tasks depending on the coordination required among the agents executing them.

Definition 2.3. Loosely-Coordination Tasks

Loosely-Coordination Tasks (LC-T) are tasks in which the agents executing them *do not require* or consider information about tasks being carried out by other agents in order to make progress on their task execution.

In a similar way, tight-coordination tasks are defined below:

Definition 2.4. Tight-Coordination Tasks

Tight-Coordination Tasks (TC-T) are tasks in which the agents executing them require or consider information about tasks carried out by other agents in order to make progress on their task execution.

Examples of Tight-Coordination tasks are:

- *DeliverObject* and *GenerateRoute* tasks. When agent A, with only *carrying* capability, executes *DeliverObject* that requires a route to be generated by agent B, which possesses the means to generate a map from start to destination points.
- CarryHeavyObject task. Assuming the object is indivisible, when agents A and B carry or push the object, the actions of agent-A directly affects the actions of agent-B and viceversa, therefore, to succeed they need to coordinate their actions.
- MoveInFormation task. When agents A and B move as a team, they need to consider the other agent's position to adjust their own position if needed.

Tight-Coordination tasks can be divided in two sub-types as defined below:

Definition 2.5. Tight Coordination-Simple Task

Tight Coordination-Simple Task (TC-ST) are simple tasks that require the joint actions of multiple agents.

Figure 2.1 depicts an example of a task tree for the Clear Site application. The mission in this application is to clear an area by removing objects of two types: ObjectsA and

ObjectsB. In this example ObjectsB are indivisible. The task tree is represented as an acyclic rooted tree with two specific relationships besides the parent-child relationship in a tree: (1) the *and* relationship that specifies that all the children in this relationship need to be achieved for the parent task to be achieved, and (2) the *triggers* relationship that specifies a way for a task t_a to create another task t_b. If the object to be removed by executing task RemoveObjectB is indivisible, but requires more than one agent to move it, then the agents assigned to an instance of this task need to coordinate their actions to successfully achieve RemoveObjectB, and for this reason RemoveObjectB is an example of a TC-ST.

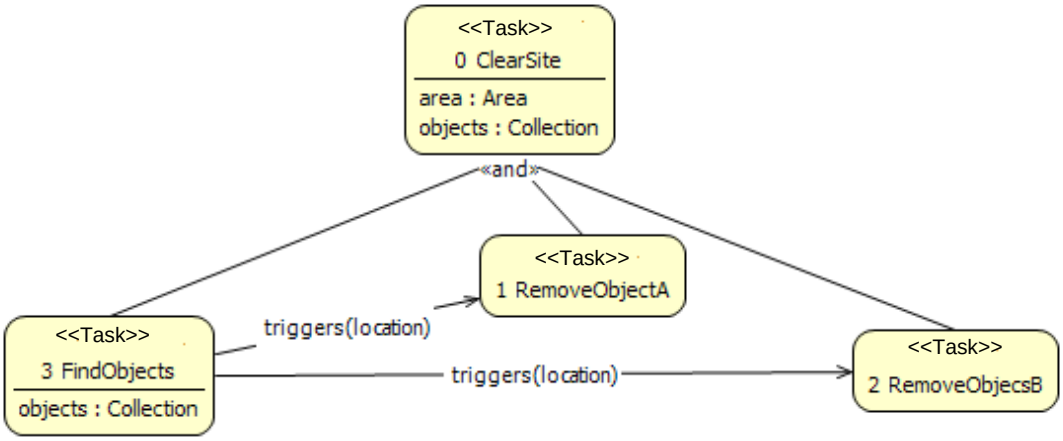

Figure 2.1: *Tight Coordination-Simple Task*

Definition 2.6. Tight Coordination-Complex Task

A *Tight Coordination-Complex Task* (TC-CT) is a parent of a TC-T.

Figure 2.2 depicts a task tree for a team of n agents that need to relocate while moving in a specific formation. The task tree is similar to the one in Figure 2.1, but this tree introduces a *precedes* relationship to indicate that task t_a needs to be achieved before task t_b can start being executed. *TeamRelocation*, *GetInFormation* and *GetToLocation* are TC-CT tasks; the tasks can be decomposed into subtasks that require coordination among the agents executing them. Each agent in the team executes *GetInPosition-i*, and if no

specific coordinates are given, the agents need to know current and changing position of other agents. Agents executing *MoveToLocation-i* need to communicate with each other, or know somehow where the teammates are while moving in formation to the destination.

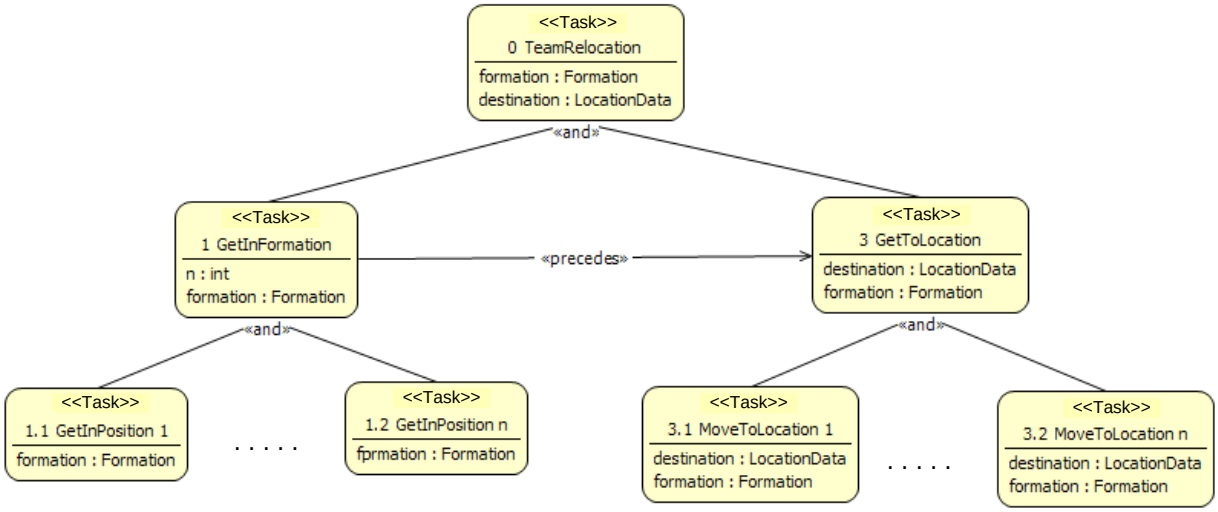

Figure 2.2: *Tight Coordination-Complex Tasks*

TC-ST (Definition 2.5) and TC-CT (Definition 2.6) specify two types of tasks addressed in this research. TC-ST are task considered in most solution proposed in the literature. TC-CT are task that are introduced in this research. This task abstraction allows the system to provide feedback at a level of groups of tasks, which is important when the user requires information about higher level tasks, such as cleaning the entire site, as in Figure 2.1, or as in the case the user finish relocating, got already in formation, or the team finished moving to a different location, like in Figure 2.1.

After presenting the types of tasks addressed in this research, Section 2.1.2 introduces the type of MAS addressed in this research and Sections 2.1.3 and 2.1.4 present the problems addressed in this research. The problems are described first as posed in other research areas such as Operational Research (OR). Later, the problem descriptions used in this research are specified by Definitions 2.10 and 2.11. The general problem statement is presented in Section 2.3.

2.1.2 Problem Description

In this section, the type of MAS considered in this proposal are presented. In this research, the Single-Task Robot, Single-Robot Task Instantaneous Assignment (ST-SR-IA), and the Single-Task Robots Multiple-Robot Tasks, Instantaneous Assignment (ST-MR-IA) Multi-Agent Systems, as defined in [28] are addressed. In other words, the Multi-Agent Systems considered are systems where agents can work on a single task at any time, tasks can require one or more agents to be executed, and the allocation algorithms do not consider future or probabilistic system information. When a task requires more than one agent, the task is specified as a complex task as defined in Definitions 2.5 and 2.6.

In the previous sections, the kind of systems and the type of tasks have been discussed. Now, the introduction of some definitions used in the *Single Agent Task* and *Multiple Agent Task Allocation* problems are presented. These definitions assume the existence of a set of agents **A**, and a set of tasks **T**.

Definition 2.7. Set of agent teams

The *set of all possible teams of agents* from A is given by the powerset of A.
Let 2^A be the set of agent teams.

Definition 2.8. Coalition

A *coalition* τ is a set of agents and $\tau \in 2^A$

Definition 2.9. Allocation

An *allocation* σ is a function that matches a task t $\in$ T to a set of agents $\tau \in 2^A$.
$\sigma : T \rightarrow 2^A$

2.1.3 Single Agent Task Allocation Problem

The single agent task allocation problem in MAS, specifically in Multi-Robot Systems, has been reduced to an instance of the Task Allocation Problem in OR [27]. An example of the task allocation problem in OR is presented in [39] as follow:

Assume there is a set of I people and a set of J jobs, and the size of the sets are the same. Each person is to be assigned exactly on job from J. There is a cost, or some other performance measurement, c_{ij} for person i assigned to job j. The problem is to find the total assignment such that the sum of the costs c_{ij} in the assignment is minimized.

The problem has also been formulated based on the Resource Constrained Project Scheduling Problem studied also in OR [40]. A brief formulation is presented here based on the one presented in [8].

Given n activities a_i i = 1,.., n, and given m renewable resources $r_k, k = 1, .., m$. A constant amount R_k units of resource r_k is available at any time. Activity a_i is executed for a period of p_i units of time, and during that period of time, a constant c_{ik} is occupied from resource r_k. Precedence constraints between activities may exist and represented by the relationship $i \rightarrow j$, which means activity f i must be completed before j can start. The challenge is to define start times S for the activities $a_i, .., a_n$ such that three conditions hold:

- *The total demand for each resource at time t is always less or equal to the amount available for that resource.*
- *The precedence constraints for the activities to be executed are fulfilled.*
- *The makespan $C_{max} = max_{i=1^n} C_i$ is minimized, where C_i wierdSymb $S_i + p_i$ is assumed to be the completion time of activity i*

As in [27], the single agent task allocation problem addressed in this research can be reduced to an instance of the task allocation problem in OR; a formalization of this problem is presented below. The problem defined in [27] is based on *jobs* and *workers*; tasks and agents are used here instead.

Definition 2.10. Single Agent Task Allocation Problem

The *Single Agent Task Allocation Problem* (SA-TAP) is:

- A prioritized set of $\boldsymbol{n}$ tasks $T = \{t_1, ..., t_n\}$, with costs $k_1...k_n$; where k_n represents the cost of executing t_n
- A set of $\boldsymbol{m}$ agents $A = \{a_1, ..., a_m\}$ each agent with a set of capabilities $C_i = \{c_1, ..., c_j\}$ for $1 \leq i \leq m$ and $j \geq 1$
- A cost function $\kappa : T \rightarrow \mathbb{R}$ where $\kappa_i(t_k)$ represents the cost of agent a_i to execute task t_k

The problem is to generate the set Φ of assignment $\sigma = < a_i, t_k >$;
where $1 \leq i \leq min(m, n)$ and $1 \leq k \leq n$ subject to minimize the global cost $\Bbbk(T)$

$$\Bbbk(T) = \sum_{\sigma \in \Phi} \kappa_i(t_k) \tag{2.1}$$

In the task allocation problem addressed in OR, the set of tasks T is static. To complete the problem reduction to an instance of the task allocation problem in OR when T changes dynamically, iterating over the solution for the static problem is necessarily, or rewriting the SAT-AP to include time and ∞ in the cost function is also possible.

Gerkey and Mataric [27] show that optimal solution for the static problem can be found in polynomial time by casting the problem to an integer linear program.

2.1.4 Multiple Agent Task Allocation Problem

The definition of the multiple agent task allocation problem is similar to the SA-TAP, except we need to replace the a_i in the assignment tuple for a subset of agents from A. The cost function needs to be defined based on the agent subset.

This problem can be divided into two subproblems: (1) generate the set of subset of agents, and (2) allocate the tasks to the agent teams.

The mathematical implications for generating the set of agent teams is presented in Section 2.1.5. What is important to mention here is that the problem of finding an optimal

allocation of tasks to agent teams is an NP-Complete problem, which generally have non-practical optimal solutions. The best approach to this problem is an approximation to the optimal solution.

Definition 2.11. Multiple Agent Task Allocation Problem The *Multiple Agent Task Allocation Problem* (MA-TAP)

- A prioritized set of $\boldsymbol{n}$ task $T = \{t_1, ..., t_n\}$, with weights $w_1 ... w_n$
- A set of $\boldsymbol{m}$ agents $A = \{a_1, ..., a_m\}$ where each agent has a set of capabilities $C_i = \{c_1, ..., c_j\}$ for $1 \leq i \leq m$ and $j \geq 1$
- An individual agent cost function $\kappa : T \rightarrow \mathbb{R}$ where $\kappa_i(t_k)$ represents the cost of agent a_i to execute task t_k
- A set of coalitions τ as described in Definition 2.8
- A multi-agent cost function $\Bbbk(t) = \sum_{a \in \tau} \kappa_i(t_k)$

The problem again is to generate the set Φ of assignment $\sigma = < \tau_i, t_k >$; where $\tau_i \in 2^A$, $1 \leq k \leq n$ and $1 \leq i \leq min(n, |2^A|)$ subject to minimize the global cost

$$\Bbbk(T) = \sum_{\tau \in 2^A} \Bbbk_i(t_k) \tag{2.2}$$

Finally, before stating the specific problems addressed in this research, a discussion of Mission, Task, and Agent representation is presented in Section 2.2.

2.2 Mission, Tasks and Agent Representations

The mission M in this research is represented by an acyclic rooted tree where t_0 is the root of the tree, and each node represents a task, see Figure 2.3. A task can be: (1)a complex task, which can be decomposed in subtasks or (2) a simple task, which is a leaf in the task tree. T represents the set of all tasks in the task tree excluding the root.

Besides the *Parent* and *Child* relationships between nodes in a tree, tasks in set T are related and constrained by the following relationships: *Conjunctive*, *Disjunctive*, *Precedes*, *Triggers*, and ¬ *Trigger*. For these relationships a similar definition as in [17] is used:

Conjunctive A task is *conjunctive* if it is achieved when all of its children are achieved.

Disjunctive A task is *disjunctive* if it is achieved when at least one of its children is achieved

Precedes *Precedence* is a relation among tasks that ensures that no agents work on a specific task until all task that precede that task have been achieved

Triggers/¬ Trigger The *triggers* relations allow one task to be created/removed by a second task when a specific event occurs

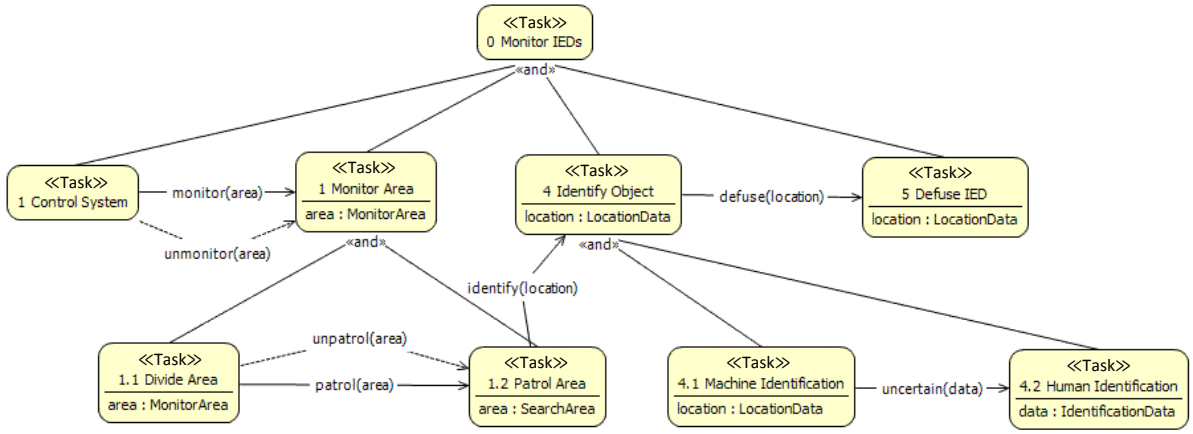

Figure 2.3: *HuRT IED Mission*

Figure 2.3 depicts the mission for the Human Robot Team IED application. The main task for the agents is to monitor a specified area for the possible presence of IEDs. The main task, MonitorIEDs, is divided into four subtasks: ControlSystem, MonitorArea, IdentifyObject, and DefuseIED. The ControlSystem task loads the information about the area where the agents are deployed, and triggers the task MonitorArea passing the specific are to

be monitored. MonitorArea is subdivided into two subtasks: DivideArea and PatrolArea. DivideArea divides the area to patrol into subareas based on the number of agents available, or a parameter specified by the human agent, and triggers as of a PatrolArea task for each subarea. If an agent detects a suspicious object while executing a PatrolArea task, it triggers the IdentifyObject task, which is is divided into two subtasks: MachineIdentification and HumanIdentification. The agent tries to identify the suspicious object by executing the MachineIdentification task, but if it fails, it requests human input by triggering the HumanIdentification task. If the suspicious object is identified as an IED, the DefuseIED task is triggered, otherwise the agent continues patrolling the area.

Not all the tasks in the mission are always ready to be allocated. The restrictive relationships precedes and triggers determine what tasks can be allocated at a given time, which are in the Ready Task set T_R.

Definition 2.12. Ready Task Set

T_R is a set of tasks such that $T_R \subseteq T$ and $\forall t \in T_R$, all preconditions to start working on t are satisfied

Each task $t_i \in T_R$ requires a set of capabilities in order to be completed, and each agent in the set A possesses a set of capabilities that allows the agent to execute a task. These two sets are specified in Definitions 2.13 and 2.14.

Definition 2.13. Required Capabilities

Let $C_{t_i} = \{c_1, ..., c_k\}$ be the set of the required capabilities for task t_i to be a achieved.

Definition 2.14. Agent's Capabilities

C_{a_j} is the set of capabilities of agent a_j.

2.3 Problem Statement

The problem to address in this dissertation is as follow:

Given a mission (M) that is divided into a set of Tasks (T) with predefined priorities and each task has an execution cost k, and having a set of agents (A) that poses some capabilities (C_a); the problem is to generate the set (Φ) of task-agent allocations in order to accomplish the mission and minimize the communication cost incurred in the allocation process.

To solve the problem, it is assumed that each agent has access to the following organization information:

Definition 2.15. Organization Knowledge

The *Organization Knowledge* (OK) available to each agent is defined as a tuple $OK = \{M, T, A, C\}$, where:

M is the mission represented by an acyclic rooted task tree, where t_0 is the root task composed by the sub tasks specified in the set T.

T $= \{t_1, ..., t_n\}$ is the set of tasks in the mission, excluding the root task.

A $= \{a_1, ..., a_m\}$ is the set of agents in the organization.

C $= \{c_1, ..., c_i\}$ is the collection of all capabilities in the organization. $C = \bigcup_{a \varepsilon A} C_a$.

When generating the output of the problem, agents consider their own cost function in executing a task, and in the case of MA-TAP, agents also consider the cost function of the coalition of agents τ.

Definition 2.16. Agent Cost Function

The *Agent Cost Function* is defined as: $\kappa : T \rightarrow \mathbb{R}$, where $\kappa_i(t_k)$ represents the cost of agent a_i to execute task t_k

Definition 2.17. Coalition Cost Function

The *Coalition Cost Function* is defined as:

$$\Bbbk_i = \boldsymbol{f}(\kappa_j) \tag{2.3}$$

where $\Bbbk_i$ is the cost function for the i^{th} coalition of agents and $1 \leq j \leq \mid \tau_i \mid$; . The $\boldsymbol{f}$ symbol represents an operation over the individual cost functions. For example, if $\boldsymbol{f}$ is the sum Σ of each cost, we have:

$$\Bbbk_i(t) = \sum_{a \in \tau_j} \kappa_j(t) \tag{2.4}$$

The output of the problem is defined as the set Φ of assignments $\tau = < r_i, t_k >$; where $r_i \in \Re$, $t_A \in T_A$, $1 \leq i \leq min(m, n)$ and $1 \leq k \leq n$ subject to minimize the global cost :

$$\Bbbk(T) = \sum_{\tau \in \Re} \Bbbk_i \tag{2.5}$$

2.4 Summary

In this section a mathematical formulation of the Task Allocation Problems in Multi-Agent System is presented. A *task* is a general abstraction used in MAS and a brief discussion of a general *task* is presented following by the definition for *simple tasks* and *complex tasks*. Tasks are then classified as *loosely* or *tightly* coordinated tasks based on the coordination required among the agents executing them. The definition for the two types of task allocation problems are addressed in this research are introduced in this chapter: (1) the Single Agent Task Allocation Problem (SAT-AP) and (2) the Multiple Agent Task Allocation Problem (MAT-AP). Also, the representation of mission, tasks, and agents are specified once the problem addressed in this research is formally presented. In the next chapter, the mathematical foundations and other technologies that support the solution proposed in this research are discussed.

Chapter 3

Background

In this chapter, the mathematical foundations and other technologies that support the DTAACS-OK framework are presented. The framework proposed in this dissertation (DTAACS-OK) addresses the problem of allocating tasks to a team of robots in a cooperative Organizational Multi-Agent Systems (OMAS). This chapter is divided in two main areas: (1) mathematics and (2) computer sciences. Other areas like game theory and constraint satisfaction, used in other solutions proposed in the literature, are discussed in Chapter 7.

3.1 Mathematical Background

In this section I present the mathematical foundation for the Single-Robot and Multi-Robot Task Allocation Problems defined in Chapter 3. Since these problems deal with groups of agents and tasks, a brief review of *Countable Sets* is presented in Section 3.1.1. Because the Multi-Robot Task Allocation Problem deals with tasks that can be assigned to a group of agents, a brief discussion on Combinatorics is presented in Section 3.1.1. The task allocation problem is about forming pairs, combining a task with an agent or agents. The goal is to find the optimal combination pair, which in this research is addressed using *Combinatorial Optimization* discussed in Section 3.1.3. Combinatorial optimization is used extensively in *Operational Research* (OR) as discussed in Section 3.1.2. OR is an area of study that models the task allocation problem and inspired the models used in this research. Due to the computational complexity (NP-Complete) of combinatorial optimization problems, a

discussion of the *Linear Programming* technique is presented in Section 3.1.4. The Multi-Agent Task Allocation problem requires generating groups of agents to work on a task, thus the *Set Partition* and *Set Coverage* problems are presented in Section 3.1.5.

3.1.1 Sets and Combinations

Before the discussion of deeper areas of mathematics related to this research, it is important to highlight that the allocation problems deal with Countable Sets, in particular Finite Sets. *Countable Sets* are defined as collections of objects in which cardinality is the same as some subset of the set of natural numbers. In this this research, a proper subset of the natural numbers, a *finite set* is required. An important set in the approach presented in this proposal is the *set of feasible solutions* to an instance of the task allocation problem. This set is required to be finite for the technique used to solve the problem.

As mentioned above, the problem addressed in this research is to fine a set of pairs formed by a task and a set of agents, which is the domain of two specific areas: *Operations Research* (OR) and *Combinatorics*. OR is discussed in Section 3.1.2, and combinatorics in Section 3.1.3.

3.1.2 Operations Research

Operations Research (OR) is the systematic effort to build and organize knowledge in pursuit of improving decision making and efficiency [53]. OR can be applied to problems from a variety of areas such as transportation, organization, and economics. A typical problem addressed in OR is the assignment of n jobs to n workers in a factory, where each worker has to be assigned to exactly one job. This is the standard *Task Allocation Problem*, which in this research is called the Single Agent Task Allocation Problem (Section 2.1.3). OR is multidisciplinary; some of the most important subject areas are mathematics, statistics, economics, psychology, physical science and sociology [39]. In this section, the mathematic aspect of OR is discussed. In particular, the quantitative models of the problem formulation

with optimization as the objective[1], which fits the goals in this research.

3.1.2.1 OR Models

In the context of this proposal, *models* are mathematical statements of the relationship between all the important factors of a problem. An OR *ideal* model includes all parts of the organization or system, although in some cases, a model of a part of the organization is beneficial (for example, when there are relatively few implications for other parts of the system). In a *normative* model, a mutually consistent decision in all the sub-problems is pursued and there is a specific parameter to be optimized. A *simulation* or *positive* model incorporates only one area of the organization or system and is used to simulate situations upon a particular decision. The models formulated in this research proposal fit the normative classification, although the models do not consider all aspects of the agent organization, such as agents' scheduler. The aspects that these models do consider are agents' availability, agents' capabilities, the tasks ready to be allocated, and tasks' capabilities requirements.

When the OR approach is used to address the task allocation problem, six phases are suggested [53]: (1) defining the problem, (2) constructing the model, (3) gathering data, (4) solving the model, (5) validating the solution, and (6) implementation. Problem definition and model construction are presented in Chapter 3 in this proposal. The rest of the proposed steps are presented in the Chapter 7. It is common that a problem have many possible valid solutions and OR aims to use combinatorial optimization to identify the best solution for the specified objective function. Combinatorial Optimization is presented in Section 3.1.3. An important and widely used technique when modeling an OR problem is Linear Programming, which is discussed in Section 3.1.4.

3.1.3 Combinatorial Optimization

Optimization is about finding the best solution to a problem. When talking about optimization, it is expected that the problems have more than one solution and that each solution

[1]Other possible objectives include prediction and control.

has a quantitative value that can be measured and compared to other solutions' values. The value can be in the form of a benefit, such as profit that can be maximized, or in terms of a loss, such as cost that can be minimized. The class of problems for which there exist a *finite* number of solutions is studied in *combinatorial optimization* from applied mathematics. The problem relevant to this research, the *Task Allocation Problem*, is part of this class of combinatorial optimization problems. The general problem of combinatorial optimization can be posed as a maximization or minimization problem as follows. Let problem p have a finite set of solutions S; assume $x \varepsilon$ S can be evaluated by a function $f(x)$ that assigns a value to solution x. The problem is to find the solution x with the maximum or minimum value. Formally defining the optimization problem requires the definition of the global maximum and global minimum of f.

Definition 3.1. Global Maximum

$x^* \varepsilon$ S is a global maximum of f if

$f(x^*) \geq f(x)$ for all $x \varepsilon$ S

Definition 3.2. Global Minimum

$x^* \varepsilon$ S is a global minimum of f if

$f(x^*) < f(x)$ for all $x \varepsilon$ S

The general *maximization* problem of combinatorial optimization is to find x^* such that x^* is a global maximum of f as defined in Definition 3.1. The general *minimization* problem of combinatorial optimization is analogous.

Finding the solution x with maximum or minimum value from a finite set of solutions might seem to be a straight-forward problem; however as discussed in Section 3.1.3.1, the solution can be intractable from a computational point of view.

3.1.3.1 The Fundamental Algorithm

Given the definition of a particular finite set S of solutions for the problem p and a function $f : S \rightarrow \Re$ that calculates a real number that indicates how good each solution $x \varepsilon$ S is, the

problem to find the solution x with maximum or minimum value can be found by following this approach:

> Calculate the value $v = f(x)$ for each $x \varepsilon$ S, compare and pick the one with the highest or lowest value, depending on being a maximization or minimization problem.

This approach is called the *Fundamental Algorithm* in combinatorial optimization. In theory, this algorithm can find the solution for a combinatorial optimization problem with a finite set of solutions. Unfortunately, for problems with a large number of solutions, the time required to find the optimal answer is not acceptable. For example, an instance of the *traveler salesman problem* with 21 cities to visit, the algorithm would need to consider 21! possible solutions that would take over 16,000 years of continuous computational calculations. For this reason, other techniques are used to find optimal and near optimal solutions to combinatorial optimization problems. One of these techniques is Linear Programming (LP). LP can find the optimal solution to some combinatorial problems, as discussed in Section 3.1.4.

3.1.4 Linear Programming

Linear Programming (LP) is an area in mathematics that is extensively used in combinatorial optimization. LP requires the objective function and all the constraints of the problem to be linear; many real world problems can be formulated in this way. The general LP problem is formulated as follows:

$Maximize\ c_1x_1 + c_2x_2 + ... + c_nx_n$

subject to

$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b_1,$

$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \leq b_2,$

. . .

$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b_m$, and

$x_1, x_2, ..., x_n \geq 0$

Problems in the general LP problem format can be converted into a *LP Standard Form*, which can be solved as described in [7]. The interesting property of problems formulated using *LP Standard Form* is that it is clear how the optimal solutions can be found in the space of feasible solutions. The detailed explanation and proof for this claim is found in [23]. The important point to highlight here is that, for optimization problems that can be modeled by LP, an optimal solution can be be found in an acceptable computational time using a technique called *Integer Programming* discussed in Section 3.1.4.1.

The Single-Task Allocation problem with a linear objective function can be formulated using LP as shown in Chapter 3.

3.1.4.1 Solution Techniques for Combinatorial Optimization Problems

Some basic techniques for solving LP problems include *Integer Programming* (IP), *Dynamic Programming* (DP), and *Heuristic* problem solving.

Integer Programming is an LP problem in which the variables are restricted to integers. Some approaches to solve an IP include enumerative techniques and cutting planes. *Branch-and-Bound Enumeration* is an enumerative technique that guarantees finding an optional solution, if one exists, to any IP problem. *Cutting-plane* is an alternative enumeration approach that is useful when variables are not integers. This method assumes that all the variables are rational; to eliminate the non-integer portion, *cuts* are introduced progressively until all the fractional parts of the feasible region are removed. After the non-integer portions are removed, the enumerative techniques of the *Simplex Method* [7, 25, 43], which is a pivoting algorithm for solving certain types of LP problems, is used to solve the problem. If the solution is an integer, then it is also optimal. Otherwise, cutting-plane defines further manipulation to solve the LP problem (See [24]).

Dynamic Programming (DP) is an alternative for solving LP problems in which decisions can be made in progressive steps. The problem can be divided into stages, each with at

least one state in which a decision can be made. To transform a state in the current stage to a state in the next stage, a decision needs to be made. The optimal decision for each of the stages does not depend on any previously decisions made, only upon the current stage and the transformation cost for that stage. This is called the *principle of optimality of dynamic programming*.

Heuristic Problem Solving. Even though, in theory, an optimal solution can be found for a problem with a finite set of feasible solutions, the computational cost can be too high to be acceptable. Unfortunately, most of the interesting problems have a large number of solutions, as illustrated both by the traveler salesman problem in Section 3.1.3.1 and the Multi-Agent Task Allocation problem addressed in this research. Thus, for problems that fall in the NP class, an algorithm that finds a near optimal solution is acceptable. These approximation algorithms use a *heuristic* function to find a solution that is not guaranteed to be optimal, but is "close enough" and can be obtained in an acceptable time frame. A good heuristic has some advantages over standard algorithms of combinatorial optimization. In addition to being able to find a near optimal solution in a acceptable time, heuristics can be flexible and easy to implement. In this research, the heuristics implemented in the algorithms are application specific, and are discussed in Chapters 7 and 8.

3.1.5 Set Partition and Set Coverage

One of the problems addressed in this research is the problem of allocating tasks to teams of agents. In an instance of this task allocation problem, more than one team of agents capable of executing a task may exist. These teams, or *coalitions*, of agents are generated from the set of agents in the organization. This section presents a brief discussion of set partition and set covering, emphasizing task allocation and its computational complexity. The partition of a set S is a collection of disjoint nonempty subsets of S, such that the union of the subsets results in the set S. The definition of set coverage is obtained by relaxing the restriction of the subsets to be disjoint. If each subset has a positive cost, finding the cover with the

minimum cost is known as the *set covering problem* and it is known to be NP-Complete [14]. The *set partition problem* is defined similarly.

In the previous sections, the mathematical background that supports the framework proposed in this work was presented. The task allocation problem can be tackled using other approaches like *game theory* and *constraint satisfaction*, which are discussed in Chapter 7. In the following sections, the areas of computer science related to this research proposal are discussed. Distributed Systems are presented in Section 3.2.1. Agents, Multi-Agent Systems (MAS), and Organization-based MAS are presented in Section 3.3. In this research, OMACS [18] is the framework used by DTAACS-OK for evaluation purposes. A discussion on OMACS [18] is presented in Section 3.4. Appendix A discusses how DTAACS-OK might be integrated to other OMAS frameworks.

3.2 Computer Science Background

An important characteristic of DTAACS-OK is that its algorithms use distributed organizational knowledge and aim to find a solution in a distributed way. The area of distributed systems is discussed in Section 3.2.1, especially with regard to the topics of *Election Protocols*, *Data Replication*, and *Concurrency Control*.

DTAACS-OK is a framework designed to be part of an Organization-based Multi-Agent System. The applicable areas from Software Engineering to be discussed are Multi-Agent Systems, in particular Organization-based Multi-Agent Systems and Multi-Robot Systems. in this proposal, DTAACS-OK is integrated into the OMAS framework OMACS [18] however there exist other OMAS frameworks that are discussed in more detail in Chapter 7.

3.2.1 Distributed Systems

As previously mentioned, DTAACS-OK approaches the Task Allocation problem in a distributed way and uses a Distributed Organization Knowledge (DOK). The DOK is replicated information each agent posses about the mission, agents and their capabilities status,

(a more detailed discussion about DOK is presented in Chapter 4). An important requirement in DTAACS-OK is that the DOK must be identical in each agent at the time the agents make a decision about the same system state; that is, if agents need to make decision d_3, the DOK must be in state s_3 in each agent by the time it reasons about making decision d_3. This requirement is achieved by ensuring that the DOK possesses the one-copy serializability property. In the current version of DTAACS-OK, only one agent can update the DOK at a given time. To determine what agent's transaction to execute next, a simple election protocol is followed. In distributed systems, the solutions proposed can be categorized as *token-based* solutions and *non-token-based* solutions. In token-based solutions the notion of token is introduced, which represents a control point. This control is passed around among the agents; the agent that possess the token is allowed to acess the shared resource [55]. In Section 3.2.1.1, non-token-based solutions are discussed, as it is the type of solution used in this proposal. Non-token-based solutions are truly distributed solutions in which all processes communicate with one another to determine which is the one to access the shared resource. Several algorithms have been proposed in the literature, and some of them are discussed below.

3.2.1.1 Election Algorithms and Mutual Exclusion

Mutual exclusion is a key problem in distributed systems. Mutual exclusion guarantees that only one process, among a set of processes, accesses a shared resource at a given time. In this proposal the agents in DTAACS-OK can only handle one transaction at the time; therefore, a mechanism to ensure this requirement is needed.

Lamport's Algorithm. This algorithm guarantees three conditions: (1) the resource is first released before granted, (2) requests are granted in the order they are received, (3) if every resource granted is released, every request is eventually granted. The rules of the protocol are as follows:

- A process interested in the shared resource sends a timestamped request to all the

other processes and adds the request to its own queue.

- Each receiving process adds the request to its queue and sends an *ack* back.
- A process holding a resource releases the shared resource by sending a release message to all other processes.
- When a release message is received, the corresponding request is removed from the queue.
- The process determines that it can access the shared resource if and only if:
  - It has a request in the queue with timestamp t, and
  - all other requests in the queue have t greater than t, and
  - it has received a message from every other process with timestamp greater than t.

Ricart and Agrawala's Algorithm [26, 55]. This algorithm is an improvement to Lamport's algorithm. It combines the functionality of the ack and release messages. An informal description of the protocol rules is as follows:

- A process, to request a shared resource, sends a time-stamp request message to all the other processes.
- When any process receives a request to share a resource from another process, one of two actions occur: (a) it sends an *okay* message if the process is not interested in the resource or if its own request has a higher times-tamp value, (b) the request is stored in a waiting queue
- To release a resource, the process sends an okay message to all processes in the waiting queue.

- A process is granted the resource when it has received an okay message to its request from every other process.

Maekawa's Algorithm [55]. In this algorithm, a process P_i trying to acquire a shared resource does not request permission from all the processes, but only from a subset R_i of them. The subsets are required to be overlapping. The selection of the subsets can vary from a centralized form where a designated process P_c is the only element in all subsets R_i, to a fully distributed form where R_i includes all processes. Mutual exclusion is guaranteed by having each process granting only one permission to a requesting process. A disadvantage of Maekawa's algorithm is that it can lead to deadlocks.

3.2.1.2 Data Replication

In DTAACS-OK the solution to the task allocation problem depends on the Organization Knowledge (OK). The OK stores information about current task allocations, agents, and their status. Any other application-specific information required by the optimization objectives needs to be stored in the OK. In a centralized approach, the OK would be stored in a single agent that would answer all requests from the rest of the agents. DTAACS-OK, however, uses a distributed approach, in which each agent possesses a copy of the OK. That is, the OK is replicated in every agent in the organization. Data replication provides advantages including robustness and fast access to data. At the same time, it introduces some challenges, two of which are *consistency* and replica *management*. To achieve data consistency in replicated data, one copy serializability is required. Replica management control handles communication failures that can lead to network partition. Some of the techniques for replica management control include Primary Site, Active Replica, and Voting which are discussed in Section 3.2.1.4.

3.2.1.3 Concurrency Control

Before addressing concurrency control, a brief discussion on *transactions* is presented. In the context of this research, a *transaction* is defined as a set of operations to be applied to the

physical data in a replicated OK. A transaction may be an update (write) or a query (read) transaction. These transactions may consist of a set of operations that must be executed in an atomic way. *Atomic execution* is the total execution of all the operations that compose a transaction, the effects of which takes place in the replica as if there were a single operation.

The goal of *concurrency control* is to provide *data consistency* in distributed replicated data. The concurrency control protocol ensures that the execution of transactions on a replicated data system is serializable. There are two main approaches for concurrency control, *optimistic* and *pessimistic* approaches. In DTAACS-OK a pessimistic approach is implemented because there is a high chance of multiple agents trying to simultaneously commit transactions in the OK. In an optimistic approach, it is expected that concurrent access to the data happens infrequently. Two common pessimistic approaches are *lock-based* and *timestamp-based* concurrency control. Lock-based mechanism are most popular [55], and in the current version of DTAACS-OK is the one implemented; because this approach does not require any clock synchronization among the agents. When using a lock-based mechanism to achieve concurrency control, lock and unlock statements are inserted in each transaction. Locking schemas can be either static or dynamic. In a static locking schema, a transaction acquires locks on all the data objects it needs before executing any action on them. In a dynamic locking schema, a transaction acquires locks on the objects it needs at different execution stages. In the current version of DTAACS-OK, the locking approach is achieved by implementing the two-phase commit protocol, which is discussed in detail in Chapter 5.

3.2.1.4 Replica Management

The main goal of a *replica management* or *replica control* protocol is to ensure that the concurrent transaction executed on the replicated data is equivalent to the execution of the transaction on non-replicated data [55]; this is known as *one copy-serializability*. Replica control is also data consistency control. The replica control algorithms ensure that different copies of the data are mutually consistent, that is, a user has the same view of the data

regardless of which copy is accessed. A significant challenge that replica control algorithms face is communication failure. Any type of communication failure of a node in the replica system may lead to a network partition. This challenge can be tackled by replica control algorithms using three different approaches: primary replica, active replica, or voting. A brief discussion of these three approaches follows:

Primary replica approach. In this approach, it is usually assumed that only node failures can occur and communication is reliable. One node is designed as *primary* and the rest as *backups*. The *read* requests are sent to the primary node and no backup nodes are involved unless the primary node fails. *Write* requests are sent to the primary node, which before updating the data, forwards the request to k other nodes, after it receives k confirmations from the backup nodes, it updates the data and returns the result to the requester. In case the primary node fails, an *election* protocol is executed to determine the new primary node.

Active replica approach. In this approach, all replicas are active simultaneously. The *read* and *write* requests are broadcast and an agrement and order properties must be satisfied before replying and updating the replicated data. A mutual consistency algorithm is integrated into this approach, such as Lamport's time-stamped mutual exclusion algorithm. Usually a weaker mutual consistency is required: after applying all the updates in a time period, all replicas must show the same values.

Voting approach. This approach is derived from the data consistency approach single-write/multiple-reads, which allows a single write but no reads, or multiple reads and no writes. This approach aims to improve fault tolerance by defining a quorum-voting. Read r and write w quorums are defined that must be met before the request is satisfied. A different flavor of this approach can be found in [2].

In DTAACS-OK, the replica management is similar to an active replica approach. In the current version of DTAACS-OK, every agent possesses an active replica; to execute a write request all agents must agree, and for read operations, each agent accesses its own copy of the data. The replica management protocol is presented in detail in Chapter 5.

In summary, DTAACS-OK relies on a replicated organization knowledge that keeps key information used by the allocation algorithms. Each agent in the organization keeps a copy of the replica, which manages concurrency by implementing an election protocol and maintains data consistency by implementing a replica control that is composed by a commitment protocol and an active replica approach that all together achieve one-copy serializability, a necessary property for the organization knowledge in this research.

In the following sections Agents, MultiAgent Systems, and especially Organization MAS are discussed in relation to task allocation and to how DTAACS-OK fits into OMACS, a specific OMAS framework.

3.3 Agents, MultiAgent Systems, and OMAS

3.3.1 Agents

Wooldridge in [54] defines *agents* as follows:

> An *agent* is a computer system that is situated in some *environment*, and that is capable of *autonomous* action in this environment in order to meet its design objectives.

Later in [54] Wooldridge extends this definition to include *intelligence*, and lists some expected capabilities of an intelligent agent:

- Reactivity
- Proactiveness
- Social ability

A similar definition is given by Ferber in [21], describing an agent as a virtual or physical entity with its own capabilities, objective or satisfaction function, and partial environment representation. These characteristics make the agents capable of:

- Interacting with the environment
- Communicating with other agents
- Partially perceiving the environment
- Potentially reproducing itself

Ferber's agent definition also specify some agent's behavioral characteristics like:

- Offering some services based on its capabilities
- Attempting to satisfy its individual objectives

In the context of this research, an *agent* is a mix of these two definitions. An agent is a virtual or physical entity with the characteristics listed in Ferber's definition, and with the *intelligence* characteristics described in Wooldridge's definition.

Ferber's definition mentions the concept of agents' interaction and Wooldridge's definition lists social ability as a characteristic of an intelligent agent. This concept of agents interacting with each other leads to the discussion of MultiAgent Systems in Section 3.3.2.

3.3.2 Multi-Agent Systems

MultiAgent Systems (MAS) is an approach proposed to address the increasing complexity and higher expectations of computational systems. Some of these expectations include easily integrating with existing systems, showing some kind of intelligence, adapting to environmental changes, being reliable and being secure. Due to higher performance and more affordable hardware, computational systems are designed and deployed in diverse areas such as real-time systems [32] that increase the complexity of the requirements mentioned above. The MAS approach aims to address all these needs and several frameworks have been developed. This section presents a definition of a MAS and the areas of MAS that have a large impact on the task allocation problem addressed in this research.

Wooldridge defines a typical Multi-Agent Systems structure in [54] as a structure containing a number of agents that interact with each other through communication. The agents are situated in an environment and can interact with it (sense and effect). There may be relations among the agents, as in relations that establish hierarchy, and agents may pursue a common goal.

Ferber in [21] defines a Multi-Agent System as a system that includes the following elements: (a) a space called *environment* and (b) a set of *passive* and *active objects* (which are agents that can modify passive objects) situated in the environment. Objects are related by defined *relationships*. *Operations* are possessed by agents in order to perceive and modify objects in the environment and are specified by *operators* that determine how the environment reacts to the agents' actions.

Multi-Agent Interactions. In MAS, an interaction happens when two or more agents are related through dynamic actions [21]. In order to classify interactions in MAS, Ferber [21] identifies three components of interactions: (a) the intentions of the agents, (b) the relationship of the agents and the resources available, and (c) the skills available to the agents to pursue their objectives. The eight types of interactions identified by Ferber [21] are: (1) independence, (2) simple collaboration, (3) obstruction, (4) coordinated collaboration, (5) pure individual competition, (6) pure collective competition, (7) individual conflict over resources, and (8) collective conflicts over resources. In this section a discussion of *(4) coordinated collaboration* is presented because it is the type of interaction that takes place in the systems in which DTAACS-OK is tested.

Cooperating, Collaborating, and Coordinating.

Coordinated collaboration is a type of interaction that occurs when agents have compatible goals, but individually lack access to sufficient resources and do not possess all the needed skills to execute a particular task. This type of interaction happens in MAS that tackle areas in which a distributed approach is needed, in particular in robot societies [21]. *Coordinated collaboration* is a complex cooperation situation in which aspects of task

allocation and coordination are combined. Malone defines coordination as [33]:

> The additional information processing performed when multiple, connected actors pursue goals that a single actor pursuing the same goals would not perform.

In MAS, actions need to be coordinated for several reasons including: (1) agents need information and the results other agents provide, (2) resources are limited, and (3) resource use must be optimized. The collaboration algorithms presented in this proposal consider these three reasons. Two of the key types of information shared by agents in DTAACS-OK are the events related to the task being executed and the status of the agents in the organization. DTAACS-OK is a general framework that can be integrated into an OMAS framework; it was designed considering application domains in which resources like communication bandwidth and energy are limited and the use of these resources needs to be optimized.

3.3.3 Organization MAS

DTAACS-OK is a framework designed to be easily integrated in any OMAS regardless of how the system was designed. In this research, the applications where DTAACS-OK was integrated were designed using the Organization Model for Adaptive Complex Systems (OMACS) framework proposed by DeLoach et al. [18]. Two of the main benefits of using OMACS are (1) OMACS uses a standard agent architecture into which DTAACS-OK fits naturally, and (2) OMACS integrates GMoDS, a powerful model and set of algorithm for managing a mission represented as a rooted tree. In the following section a brief description of this model is presented.

3.4 OMACS

OMACS [18] is a metamodel for artificial organizations. It allows the design of MAS with an extended concept of an artificial organization. OMACS extends the general concept of an organization, which includes a set of agents, the roles agents play, and the relationships

among them, by adding the concepts of goals, capabilities, assignments, policies and a domain model [18]. One of the core elements of OMACS is the organization metamodel depicted in Figure 3.1. An organization is composed by Agents (A) that are capable of playing a Role (R), and by doing so, they may achieve a Goal (G). Other entities in the metamodel are the Capabilities (C) that are required to play a Role and that are possessed by the Agents. The OMACS metamodel defines two more entities; these are the Domain (D) and a set of Policies (P) that constraints the Organization. The model also defines some functions that help in the organization's performance analysis and in the assignment of goals to the agents. These functions are as follows:

oaf function $P(G \times R \times A) \rightarrow [0..\infty]$ that defines the quality of a proposed set of assignments

achieves function $(G \times R) \rightarrow [0..1]$ that defines how well a role achieves a goal

capable function $(A \times R) \rightarrow [0..1]$ that defines how well an agent can play a role

requires function $R \rightarrow P(C)$ that defines the set of capabilities required to play a role

possesses function $(A \times C) \rightarrow [0..1]$ that defines the quality of an agent's capability

potential function $(A \times R \times G) \rightarrow [0..1]$ that defines how well an agent can play a role to achieve a goal

The applications implemented to test DTAACS-OK were designed using OMACS. To integrate DTAACS-OK into these applications, a layer was developed. The main functionality of this layer was to generate a task as expected from DTAACS-OK from the goals and roles defined by OMACS. The details of the integration is presented in Section A of this proposal.

3.5 Conclusion

In this chapter, the mathematical foundations and other technologies that support the design of DTAACS-OK and the algorithms implemented was presented. Section 3.1 presents

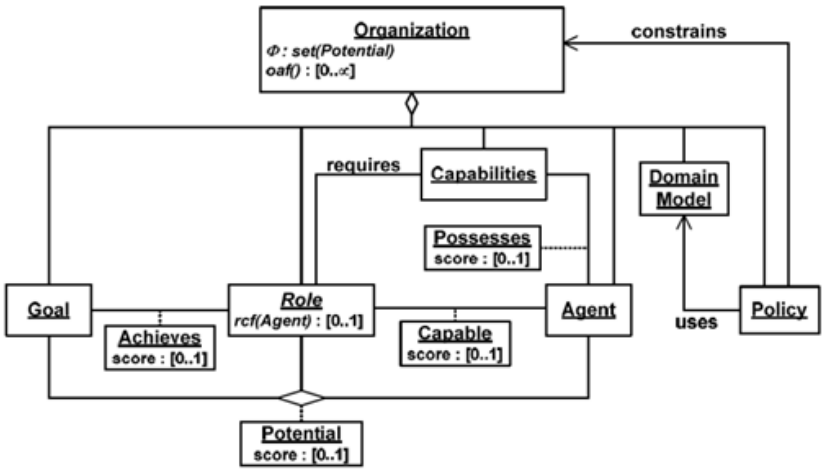

Figure 3.1: *OMACS Metamodel*

a discussion on sets, combinations, combinatorial optimizations, some techniques utilized in OR such as LP, and set partition and set covering problems. In Section 3.2 some areas of computer science such as distributed systems, Multi-Agent Systems and OMAS are discussed. DTAACS-OK tackles the task allocation problem in a distributed way and a key component that the algorithms use is the organization knowledge. Important areas from distributed systems include election and mutual exclusion algorithms, concurrency control and replica management, which are discussed in this section.

In the next two chapters, the DTAACS-OK framework is presented. In Chapter 4, the components of DTAACS-OK are discussed and the algorithms to tackle the SA-TAP are presented. In Chapter 5, the algorithms to tackle the MA-TAP are discussed.

Chapter 4

DTAACS-OK

In this chapter, a solution to the SA-TAP and MA-TAP in cooperative mission achievement in OMAS is presented. The solution is first described as a general framework giving an overview of its components and their general functionality. Later each component is discussed in detail and is illustrated with examples.

It is generally accepted in the literature that in some application domains a solution using the MAS approach is more appropriate than a single agent system. Even application domains that do not strictly require a MAS, can benefit from it [48]. MAS introduce other challenges, besides the application specific requirements and constraints, that need special attention and have been active research topics over the last few decades. MAS are usually designed to tackle missions in which the agents need to cooperate and sometimes work on the same task to achieve it. To address the complexity introduced by having a system with multiple agents, the Organization-based MAS (OMAS) paradigm has been proposed. OMAS is recommended for complex systems with multiple tasks or goals. Even for a single task mission, we have to select the most suitable agent for that task. Usually, the problem we are trying to solve can be decomposed into sub-problems or sub-tasks making the challenge more interesting. Now the selection of the agent or agents for the available tasks is more dynamic and the initial assignment can change over the life of the tasks. Several solutions are proposed that use market based or gaming techniques [15, 19, 41, 56]. Unfortunately besides the communication required to allocate tasks, these approaches also incur a heavy

communication load to keep the agents informed about the status of the mission and agents. To take advantage of the information sharing among agents, I propose a solution called Distributed Task Allocation in Adaptive Computational Systems based on Organization Knowledge (DTAACS-OK) to address the task-agent(s) matching problem.

4.1 DTAACS-OK Components

DTAACS-OK is a framework designed to address the agent-task matching problem in complex systems that adapt to dynamic changes within the system and in the environment. DTAACS-OK integrates four main components: 1) *Distributed Transaction Component*, 2) *Distributed Organization Knowledge Component*, 3) *Distributed Task Allocation Component*, and 4) *Local Agent's Information Component*. The four components, their relationships and data flow among them is depicted in Figure 4.1. The Distributed Transaction Component's main purpose is to ensure the one-copy serializability property of the organization knowledge. The Distributed Organization Knowledge Component stores and updates the organization information, which is passed to the Distributed Task Allocation Component to generate the assignments. The Local Agent's Information Component stores the current agent's information that is used as part of the organization knowledge update. The components are specified and designed to provide near optimal solutions to the task allocation problem in a multi-agent systems in different application domains. An overview of typical process is described in Section 4.1.1.

4.1.1 A General Overview of Mission Execution

We describe the general steps in DTAACS-OK using a mission $\boldsymbol{M}$ as example.

1. All agents in the organization are initialized with an identical organization knowledge. It contains the tasks in the mission $\boldsymbol{M}$, and the information of all agents that conform the organization.

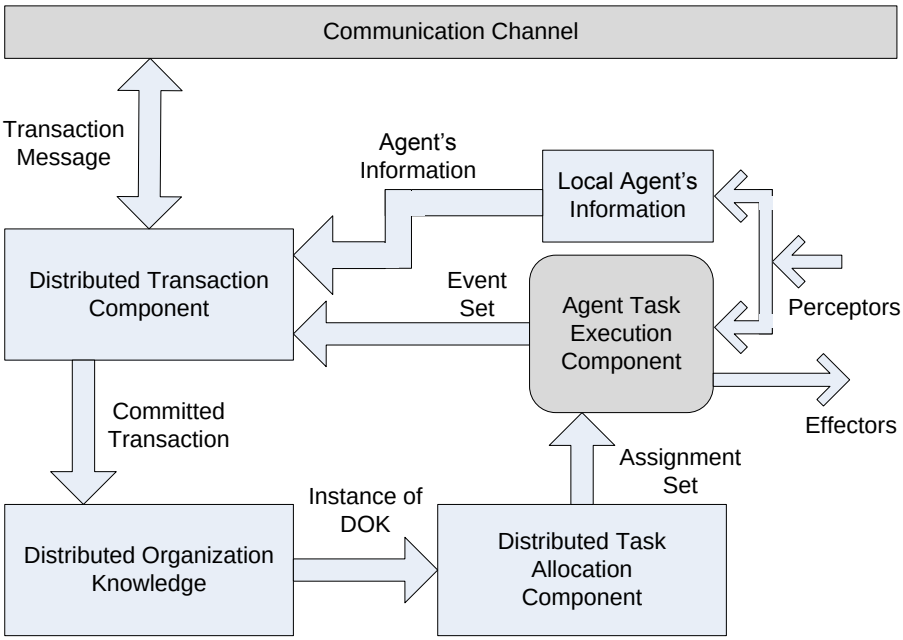

Figure 4.1: *DTAACS-OK Components*

2. The first action occurs in the Distributed Transaction Component when each agent generates the *initial-committed-transaction* that is processed by the Distributed Organization Knowledge Component to generate the first task(s) to be assigned.

3. The Distributed Organization Knowledge Component receives and processes the committed transaction. When new tasks are created, the component passes the task(s) ready to be allocated, the agents in the organization, and their capabilities to the Distributed Task Allocation Component.

4. The Distributed Task Allocation Component identifies the most suitable agent(s) for the unassigned task(s). If the most suitable agent is itself, it passes the assignment to the agent's execution component and goes to a waiting state. If there is not a new assignment for the agent, it also goes to the waiting state for a new set of task to allocate after a transaction is committed.

5. The agent's task execution component schedules the execution of the task(s) according with the application specific scheduler. At this point, any assignment set and coalition set updates are executed.

6. The agent generates an event according with the task execution, like TASK_ACHIEVED, TASK_TRIGGER, etc., passing the event(s) to the Transaction Generator module of the Distributed Transaction Component.

7. The Distributed Transaction Component starts the protocol to commit the new transaction and once consensus is achieved, it commits the transaction to the Distributed Organization Knowledge.

8. Execution continuous at step three until the mission is achieved.

4.2 Distributed Transaction Component

There are several distributed solutions for the task allocation problem that exploit the benefits of a market economy approach [15, 42]. In these solutions, task allocation ooccurs by having the agents calculate their own cost and bid for tasks against other interested agents. The task is allocated to the winner agent by the bid coordinator. There are other approaches that apply game theory to the task allocation problem [3, 12]. Chapman et.al. [12] use a distributed stochastic algorithm to solve an approximation of Markov games to define the utility function of each agent. The agents use this utility function to calculate the *cost* of executing a task, but the allocation of the tasks still involves a bidding process.

In DTAACS-OK, the information sharing that happens in a cooperative or collaborative multi-agent systems is exploited. In most cooperative multi-agent system, agents reason about other agents' task execution and internal state, while pursuing a common task. To achieve a near optimal solution, it is essential to share information about the status of tasks and agents, such as task achievement, task failure, task creation, and agent status (specifically the agent capability status). In [40], to keep the system consistent, the robots

broadcast messages to notify other robots about the tasks achieved, new tasks discovered, task execution status, task achievement, task cancelation, and task invalidation (which is generated when more agents are required to execute a task).

DTAACS-OK takes advantage of this information sharing and provides *synchronized* organization knowledge to all agents in the organization to enable them to make identical decisions using the same task allocation algorithms. Therefore, an important piece of DTAACS-OK is the Distributed Transaction Component.

4.2.1 Distributed Transaction Component

The Distributed Transaction Component (DTC) provides the mechanisms to coordinate the transaction generated while executing the mission. While coordinating the transaction, the DTC aims to guarantee the one-copy serializability property (see Section SecDistDatabaseSys) of the Distributed Organization Knowledge . The DTC interacts with the system communication layer and other three components: The *Distributed Organization Knowledge Component*, the *Agent Task Execution Component* and the *Agent's Local Information Component*. The Distributed Transaction Component includes two modules, the *Transaction Generator* and *Transaction Manager* as shown in Figure 4.2. The DTC requires the agent to generate events to inform the Transaction Generator Module about task execution and agent statuses. The Transaction Generator Module receives these events and proceeds to generate a new transaction, which is passed to the Transaction Manager. When the Transaction Manager receives a transaction, it establishes communication with the agents in the organization following a distributed transaction commitment protocol to ensure the one-copy serializability property of the Distributed Organization Knowledge. In this research, a simplified version of the Two Phase Commit protocol (2PC) is used. The 2PC protocol is one of the mechanism that helps provides the one-copy serializability property to the organization knowledge. The following section presents more detailed information about the two modules that make up the Distributed Transaction Component.

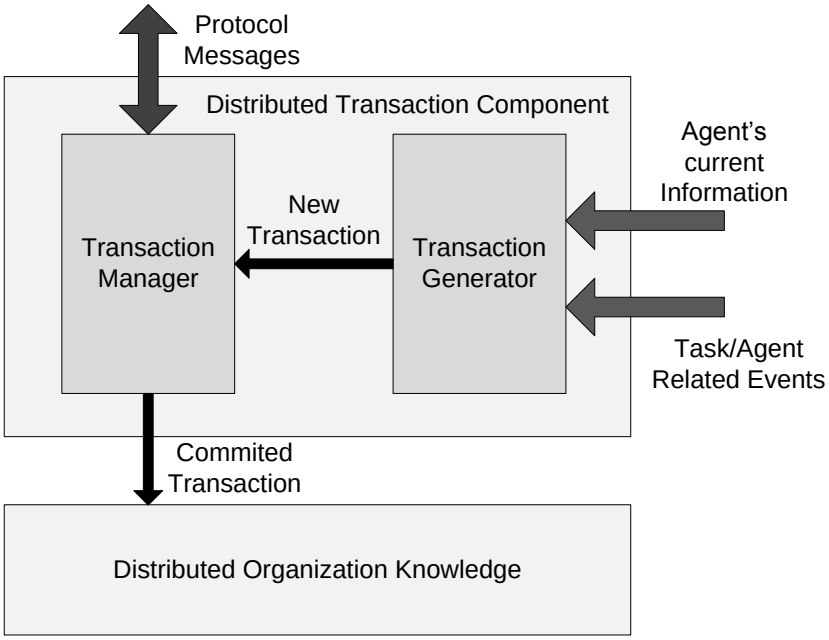

Figure 4.2: *Distributed Transaction Component Diagram*

4.2.2 Transaction Generator

The Transaction Generator is part of the *DTC* and acts as an interface between the *Agent Task Execution Component* and the *DTC*. It also interacts with the *Local Agent's Information Component* and the *Transaction Manager*, which is also part of the *DTC*. Figure 4.3 depicts the states for the *Transaction Generator*. After initialization, it waits for input events, (See Table 4.1) from the *Agent Task Execution Component*; when it receives one of these events, it proceeds to gather the latest agent information from the *Local Agent's Information Component*, (e.g. location, capability status, etc.), to include in the transaction it will create. Once the new transaction is ready, the *Transaction Generator* passes it to the *Transaction Manager* and goes back to the waiting state, (See Figure 4.4 for an example of a transaction).

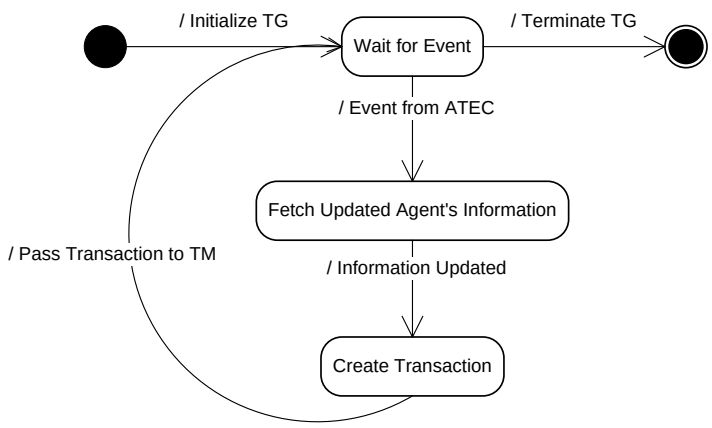

Figure 4.3: *Transaction Generator State Diagram*

Transaction		
Event	Parameters	
TASK ACHIEVED	TASK_ID	AGENT'S_CURRENT_INFO

Figure 4.4: *Transaction Example*

4.2.3 Transaction Manager

As mentioned in Section 4.2, all agents in the organization need to make identical decisions when assigning tasks to agents. Therefore, the agents need to run the same task allocation algorithm with identical input each time task allocation is required. That is, the organization knowledge, in every agent in the organization, needs to be identical at each task allocation decision point. Therefore, the organization knowledge is replicated in all agents in the organization. Replicated organization knowledge means that there are several physical copies of the same logical data in different places [2]. In our case, we maintain copies of the organization knowledge in all agents in the organization. Some of the conditions and requirements in replicated knowledge systems, including DTAACS-OK, are presented in Sections 4.2.3.1 and 4.2.3.2.

No	Event Type	Description
1	TASK_ACHIEVED	After the agent executes and completes with a successful status a task, it notifies the Task Generator for a new transaction to be created
2	TASK_FAILED	An agent that cannot complete the execution of a task for any reason, notifies the Task Generator for a new transaction to be created
3	TASK_TRIGGERED	An agent, while executing a task, fulfills the preconditions of a trigger relationship between two task, sends this event to the Task Generator
4	TASK_NEGATIVE_TRIGGERED	Similar to a TASK_TRIGGERED transaction. An agent, while executing a task, fulfills the preconditions of a negative-trigger relationship between two task, sends this message to the Transaction Generator
5	AGENT_FAILURE	The agent that detects its own or other agent's failure, notifies the Task Generator

Table 4.1: *Required Events From Agent Task Execution Component*

Before starting to describe the Transaction Manager, a definition is presented:

Definition 4.1. Decision Point.

A *Decision Point* is a state S_{dp} in the system that occurs while pursuing the mission and an allocation or re-allocation of a task decision is made.

There are several events that lead to a decision point, basically there is a decision point for each event described in Table 4.2 that the agent generates.

Claim 4.1. Enough Decision Points.

The decision points listed in Table 4.2 are sufficient to ensure progress and termination in pursuing the mission.

This claim's proof to be included in final version of dissertation.

4.2.3.1 Distributed Knowledge Systems

Distributed knowledge systems, and specifically systems using information replication, aim to achieve a level of robustness so data is always accessible, even in the presence of problems like network partitions and repository failures. Data replication can also provide faster access to the information since the data resides closer to the client. Two basic operations can be applied to the replicated knowledge, a *write-operation* to update the physical copy of the logical data, and a *read-operation* to retrieve the latest updated copy of the physical data. The challenge in any series of read and write operations to replicated knowledge is that, the replicated knowledge behavior as observed from the outside, should be the same as if the read/write operations were performed in a non-replicated knowledge system. This is a property known as *one-copy serializability* [55], and is the main goal of any *replica data management protocol*. Usually, a replica data management protocol is decomposed into two parts, the Transaction Atomicity Protocol, and the Replica Control Protocol. The former ensures the serializability of the update operation in the database, but it is not sufficient to provide one-copy serializability, which is why a Replica Control Protocol is required. In the following section a discussion of some of the proposed protocols in the literature for both parts, and the ones used in DTAACS-OK Transaction Manager are presented.

4.2.3.2 Replica Control

The Replica Control Protocol defines how the logical data is replicated, what replicas are updated when a write operation is executed, and what repository to access when a read operation is executed. There are several solutions to this problem that can be categorized based on the way data is replicated, how data is updated, and on the way the system recovers from failure [10]. Two of DTAACS-OK objectives while allocating tasks are to (1) minimize re-work or waste of effort, and (2) avoid task starvation. Both of these goals require our approach to maintain as many replicas as agents in the organization, and for the replicas to be synchronized when an new assignment is needed. According to the taxonomy

presented in [10], the Replica Control Protocol required for DTAACS-OK is Identical copies, Synchronous-all/Synchronous-available. As defined by Ceri et al. [10] these terms are defined as follow:

- *Identical copies* means that all the copies of the replicated data have the same rights, properties, and are treated the same way.
- *Synchronous-all* means that all the copies of the replicated data are updated synchronously (atomicity can be guaranteed by using the two-phase commit protocol).
- *Synchronous-available* is an invariant that the replicated data possesses, which in this case means that all available replicas are up-to-date. By assuming no network partition, there is only one possible subset of available copies at a time.

The organization in DTAACS-OK is a flat organization and all replicas represent a truly distributed replicated knowledge. Depending on the application domain and the user requirements, the Replica Control Protocol can be more flexible and allow some inconsistency in the knowledge state in case of a network partition. This flexibility provides better system performance at the cost of extra computation to synchronize and restore the organization knowledge to a synchronized state. At the time of writing this research proposal, only total replication is implemented. The experiments presented in this research proposal do not support network partition or agent failure; these features will be specified, implemented, and tested in the future.

4.2.3.3 Transaction Atomicity

Transaction Atomicity is provided by the Transaction Manager by running an atomic commitment protocol. There are several protocols, such as Two Phase Commit (2PC) protocol, Three Phase Commit (3PC) protocol, Dynamic Two Phase Commitment (D2PC) protocol, and Emulated 2PC (E2PC) protocol, that provide atomicity in the transactions and depending on the robustness, speed, and other system characteristic, one can pick the most

appropriate. These protocols define a coordinator and participants for each transaction. It is also possible to design and implement systems where more than one transaction is handled by having more than one coordinator running in parallel. In this research, an implementation of a simplified version of the 2PC protocol is used (Algorithm 1) - no transaction logging is kept-. This 2PC simplified version allows to provide the required atomicity, the limitation introduced is the ability to rollback transactions in case of network partitions.

Algorithm 1 DTAACS-OK 2PC Coordinator
input : Message m
output: Transaction status: succeeded/failed

```
1: Coordinator
2: State Q:
3: broadcast(VotingRequest)
4: GoTo W
5: State W:
6: if timeout then
7:    resendMsgTo(AllPendingCohorts)
8: end if
9: if all_ACK then
10:   broadcast(CommitTransaction)
11:   GoTo COMMIT
12: end if
13: State COMMIT:
14: WaitForACK()
15: if timeout then
16:   resendMsgTo(AllPendingCohorts)
17: end if
18: if All_ACK then
19:   commitTransaction(m)
20:   GoTo DONE
21: end if
22: State DONE:
23: return status
```

DTAACS-OK 2PC Messgae Complexity Analysis

Message Complexity:

When no communication problems is $2 + 2(n-1)$ which is $O(n)$

When communication fails with a probability $\boldsymbol{p}$, see Formula 4.2, which is also in $O(n)$

Algorithm 2 DTAACS-OK 2PC Cohort
input : Message m
output: Transaction status: succeeded/failed

```
1: Cohort
2: State Q:
3: send(vote)
4: GoTo W
5: State W:
6: waitForCommitMsg
7: GoTo COMMIT
8: State COMMIT:
9: Commit(m)
10: send(ACK)
11: GoTo DONE
12: State DONE:
13: return status
```

Reasoning: The 2PC algorithm is analyzed for the cases when there is no communication problems, and when the communication may fail depending in a certain probability. When the communication conditions are ideal, the *Coordinator* broadcast a message (line 3) to ***n-1*** cohorts (n is the number of agents in the system), the coordinator then receives $n-1$ ACK messages (line 6), broadcast the message to inform the cohorts to commit (line 10), and receives $n-1$ ACK messages. Therefore, there are $2+2(n-1)$ messages. In the case the communication may fail with a certain probability, the Coordinator will re-broadcast the messages, and the cohorts will re-send the voting and Ack messages $(n-1)$.
Let $\boldsymbol{m}$ be the total messages sent when there is no adversity, then

$$m = 2 + 2(n-1) \tag{4.1}$$

In case the comunication will fail with a probability $\boldsymbol{p}$, the total of messages will be given by: $m + pm + p^2m + ... + p^tm$ while $p^tm \geq 1$. Therefore, the total messages sent $\boldsymbol{M}$ will be given by:

$$M = \sum_{t=0}^{\log(m)/log(p)} p^t m \tag{4.2}$$

Equation 4.2 is a function on $\boldsymbol{m}$, which is $2 + 2(n-1)$, which still is a function on the number of agents in the system, therefore is also in $O(n)$.

In the current DTAACS-OK implementation, for simplicity, only one transaction is negotiated to be committed at at a time, that is, there is only one transaction coordinator at any time and the rest are participants. This implementation required a mechanism to ensure only one transaction a the time, which can be provided by a *Election Protocol*. In the current state of DTAACS-OK implementation, a simple election protocol is used, similar to the 2PC.

4.3 Distributed Organization Knowledge Component

The Distributed Organization Knowledge Component (DOK) contains the organization information that DTAACS-OK specifies as needed to make the task-agent matching. This component interacts with the Distributed Transaction Manager and the Distributed Task Allocation Component. The Distributed Organization Knowledge includes two modules, the Task Set Selection Module and the Organization Information Module as depicted in Figure 4.5. When the Organization Knowledge Component receives a *committed transaction* from the Distributed Transaction Manager, it proceeds to update the information and generate a snapshot of the organization knowledge. This snapshot is the input to the task-agent matching algorithms that the agents execute.

4.3.1 Task Set Selection Module

The main data structure in the Task Set Selection Module is the Task Tree that represents the mission. The Task Tree is a directed rooted acyclic graph that integrates all the task

No	Transaction Type	Description
1	TASK_ACHIEVED	After the agent executes and completes with a successful status a task, it notifies the rest of the agents in the organization, and sends any updated agent s information
2	TASK_FAILED	An agent that cannot complete the execution of a task for any reason, notifies the rest of the agents in the organization by sending a failure message along with the updated task and agent information
3	TASK_DEASSIGMENT	An agent, while executing a task, can get an assignment with a task with higher priority or that yields higher benefit to the organization; it releases the current task being executed to be re-allocated
4	TASK_TRIGGERED	An agent, while executing a task, fulfills the preconditions of a trigger relationship between two task, sends this message to the team for the update of their organization knowledge
5	TASK_NEGATIVE_TRIGGERED	Similar to a TASK_TRIGGERED transaction. An agent, while executing a task, fulfills the preconditions of a negative-trigger relationship between two task, sends this message to the team for the update of their organization knowledge
6	AGENT_FAILURE	The agent that detects its own or other agent's failure, notifies the rest of the agents for them to update its Organization Knowledge

Table 4.2: *DTAACS-OK Transaction Types*

(ST and CT) in the mission and their relationships. The other entities are sets that store information about the tasks status. We define the following sets:

1. Active Task Set. The set of tasks for which all preconditions are satisfied. This is the set of tasks the task allocation algorithms may assign to agents.

2. Achieved Task Set. The set of of all tasks that have been successfully achieved.

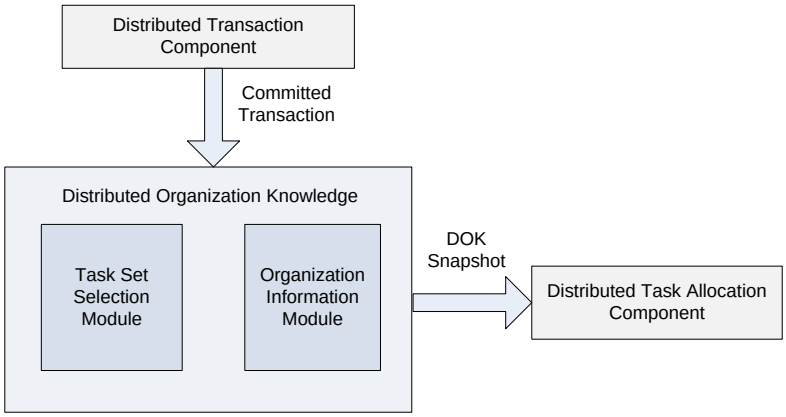

Figure 4.5: *Distributed Organization Knowledge Component*

3. Removed Task Set. The set of tasks that have been removed from the mission and are not required to be assigned.

4. Failed Task Set. The set of tasks that agents could not achieve and will not be assigned again.

When the DOK receives a committed transaction, it processes the event in the transaction and updates the above sets accordingly. For instance, if the event indicates that task t_3 has been achieved, t_3 is placed in the Achieved Task Set, and if t_3 triggers other tasks or has a precedence relationship with other tasks, the appropriated tasks are placed in the Active Task Set.

4.3.2 Organization Information Module

The Organization Information Module consists of two sets that represent the organization information required to make the task-agent matching decision. The sets representing the organization are the Agent Set and Capability Set. The Agent Set stores information about all the agents in the organization and consist of the agent's unique identifier, agent status, agent capabilities, and agent capability status, as depicted in Table 4.3. The Capability Set

stores information about all the capabilities the agents contribute to the organization, which consists of the capability unique identifier, capability name, and the agents that posses this capability.

No	Attribute	Description
1	agentID	Unique identifier in the organization that refers to a single agent
2	agentStatus	Status identifier that represents the current state of the agent. (See Table 4.4 for status information)
3	agentCapabilities	The set of capabilities the agent possesses and consist of a tuple $< capabilityID, capabilityStatus >$

Table 4.3: *Attributes representing an Agent in the Organization*

Status	Description
ACTIVE	Indicates that the agent can be considered for task allocation
FAILED	No longer capable of executing a task

Table 4.4: *Agent Status*

No	Field	Description
1	capabilityID	Unique identifier in the organization that refers to a single capability
2	capabilityName	A descriptive name
3	capabilityAgents	The set of agents that possess this capability

Table 4.5: *Capabilities in the Organization*

4.4 Distributed Task Allocation Component

The main goal of the Distributed Task Allocation Component (DTAC) is to generate a new assignment set using the latest organization knowledge. DTAC interacts with the Organization Knowledge Component and the Agent's Task Execution Component as depicted in

Figure 4.6. The information that is passed from the Distributed Organization Knowledge Component to the Task Allocation Component contains the latest snapshot of the organization knowledge that is used to update the utility criteria entities and generate the new assignment set.

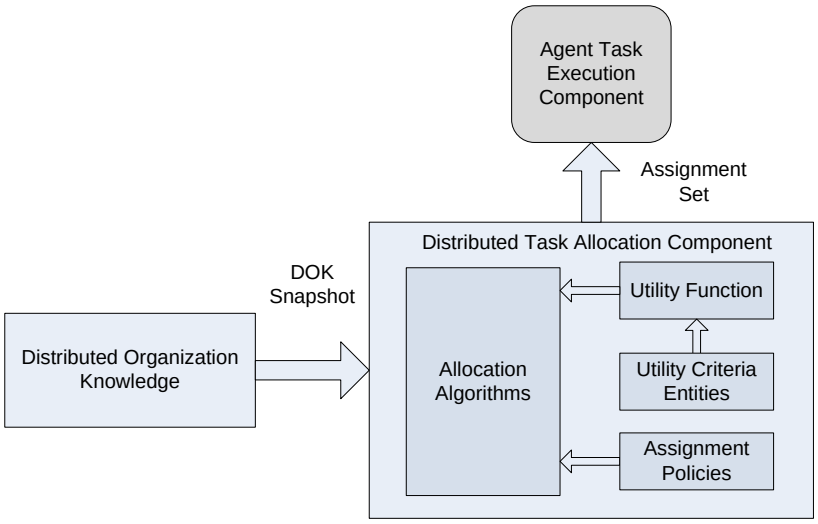

Figure 4.6: *Distributed Task Allocation Component*

This component integrates four main modules: 1) Allocation Algorithm, 2) Utility Function, 3) Utility Criteria Entity, and 4) Allocation Policies. The Utility Function is application dependent, and the assignment policies can be defined based on the optimization objectives.

4.4.1 Allocation Algorithms

4.4.1.1 WorkInMission Algorithm

This section defines the *WorkInMission* algorithm as specified in Algorithm 1. The *WorkInMission* algorithm is the main algorithm in the framework and monitors the execution of the system in order to achieve the mission. This algorithm loops while there are tasks to be assigned and the mission is still feasible. When the mission starts, each robot has the same knowledge. The algorithm starts by setting the *status* of the mission as failed and

the exit loop variable *done* as false (lines 1 and 2). The algorithm loops while there are tasks to be assigned and the mission is not completed (line 5). In line 6, a new assignment set is obtained by calling the AllocateTask algorithm passing three parameters: (a) the current task ready set, (b) the current organization agents, and (c) the agent's capabilities information. After generating a new assignment set, the algorithm determines if there is a new assignment for the agent by searching the new generated assignment set (line 7). The following scenarios are possible:

1. If there is assignments for the agent (line 8), then the tasks are sent to the Task Execution Component (line 9).

2. There is no new assignment for the agent. The algorithm then checks if the mission is achieved (line 13).

The task execution generates events (See Table 4.1) that are processed by sending them to the DTC (line 10), and the event set is set to empty (line 11). After the mission is completed (line 13), the status (success or failed) on how the mission was terminated is retrieved (line 15) and returns this information (line 16) to the caller algorithm.

Algorithm WorkInMission Complexity Analysis

Time Complexity: $O(n^4)$

Reasoning: The algorithm begins by initializing the status of the mission and a flag to determine if the mission is achieved (line 1 and 2), both run in O(1). Similarly, setting the task set to empty (line 3) runs in O(1). The while loop in line 4 iterates while the mission is not completed, which is dependent on the number of tasks that are active. Therefore, the loop runs in O(n). Setting the active task set (line 5) runs in O(1), however generating the assignment set (line 6) runs in $O(n^3)$ (See Algorithm 4. Getting the task set for a particular agent (line 7) runs in O(n), and processing the outcome events (line 10) runs in O(n) as well. Therefore, the *WorkInMission* algorithm runs in $O(n^4)$.

Algorithm 3 WorkInMission for agent a_i
input : Organization Knowledge OK
output: Mission status: succeeded/failed

```
1: status ← failed
2: done ← false
3: myTasks ← empty
4: while not done do
5:    T_A ← OK.activeTasksSet()
6:    assignmentSet ← AllocateTask(T_A, OK.agentSet(), OK.capabilitySet())
7:    myTasks ← assignmentSet.myTasks(agentID)
8:    if myTasks not empty then
9:       events ← workInTask(myTasks)
10:      processEvents(events)
11:      events ← empty
12:   end if
13:   done ← ok.missionAcomplished()
14: end while
15: status ← ok.getStatus();
16: return status
```

4.4.1.2 AllocateTasks Algorithm

The *AllocateTasks* algorithm (Algorithm 4) generates the assignment set for the tasks that are ready to be assigned and using the agents in the organization. First, the *AllocateTasks* algorithm initializes the assignment set to empty, the number of agents required for a task to one, and the current coalition to empty set (lines 1-3). After initialization, the algorithm tries to find an agent for each task in T_A and generates the assignment set (line 4). After getting the first task (line 5), the required capabilities and the number of agents for that task are obtained (line 6 and 7). If more than one agent is required by the task, the GetBestCoalition algorithm is called (line 9) (the GetBestCoalition is discussed in Chapter 5). If one agent is required, then the algorithm applies any allocation policies (line 11) to the set of agents. After filtering the agents, then the GetBestAgent algorithm is called (line 12) to determine the most suitable agent based on the list of required capabilities. It is assumed that for each task, there is at least one agent capable of executing it. In line 14, the new assignment is added to the assignment set. Once all tasks are assigned, the assignment set

is returned (line 16).

Algorithm 4 AllocateTasks in T_A
input : Active task set T_A, agent set A, and capabilities set R
output: The allocation set Φ such that for each task $t_i \in T_A$ the pair (t_i, c_j) is added to Φ and c_j is most suitable coalition for t_i
c_j contains at least one agent

```
1: Φ ← ∅
2: reqNumAgents ← 1
3: bestCoalition ← nil
4: while T_A ≠ ∅ do
5:    t ← T_A.removeFirst()
6:    capList ← t.getReqCapabilities()
7:    reqNumAgents ← t.getReqNumAgents()
8:    if reqNumAgents > 1 then
9:       bestCoalition ← getBestCoalition(A, t_i) {See Chapter 5}
10:   else
11:      A ← applyAllocationPolicies(A)
12:      bestCoalition ← getBestAgent(A, capList)
13:   end if
14:   Φ.add(t, bestCoalition)
15: end while
16: return Φ
```

Algorithm AllocateTasks Complexity Analysis

Time Complexity: $O(n^3)$

Reasoning: The algorithm begins by initializing the allocation set to nil, the required number of agents to one, and the best coalition to nil (line 1, 2 and 3) which each run in O(1). The *while* loop iterates over the task set (line 4), it removes the first task saving it to a temporary variable (line 5) which runs in O(1). Setting the required capabilities of the task list and the required number of agents (line 6 and 7) also runs in O(1). Determining the best coalition to use (line 9) runs in O(n). Applying the allocation policies to the set of agents and then determining the best agent to use for a given capability set (line 11 and 12) both run in $O(n^2)$. Adding the assignment (task, coalition) to the assignment set runs in O(1). Therefore, the *AllocateTasks* algorithm runs in $O(n^3)$.

4.4.1.3 GetBestAgent Algorithm

The GetBestAgent algorithm (Algorithm 5) determines the agent most suitable for a given task. The GetBestAgent algorithm has two parts. First, it determines who has the capabilities in *capList* and second, after finding these candidate agents, it determines who gets the assignment by choosing the agent with the best score based on the list of capabilities. The algorithm starts by initializing the best agent as nil (line 1) and the set of candidate agents to the set received as parameter (line 2). The algorithm then enters a loop (line 3) to check each capability in the list (line 4) against the agents from the candidate agents set that posses the capability, which is done in an embedded loop (line 7). In the embedded loop, the first agent is extracted (line 8), and if the agent does not posses the capability (line 9), it is removed from the list of candidate agents (line 10). After the first loop terminates, the set of candidate agents stores the agents that posses all the list of capabilities. The second part starts by initializing the score to minus one (line 15). The algorithm then enters a loop (line 16) to calculate the score of each candidate agent based on the list of capabilities (lines 17 and 18). The previous calculated score and the current score are compared to keep the highest one and to store the associated agent (Lines 19, 20 and 21). If the previous score and the current score are the same, the algorithm breaks the tie using the agents' id (lines 22 to 25). Once the loop checks all the agents, the agent identified with the highest score is returned.

Algorithm GetBestAgent Complexity Analysis

Time Complexity: $O(n^2)$

Reasoning: The algorithm begins by initializing the agent to be returned to nil and sets the agents provided to a temporary set (line 1 and 2); both of which run in O(1). The while loop in line 3 is the main loop in the algorithm and it has two inner while loops. In line 4, the first capability of the set of capabilities provided is removed (line 4), and the number of agents is stored in a variable (line 5), these two lines run in O(1). The first inner loop (line 7) gets the first agent from the list (line 8) and checks if the agent possesses a capability.

Algorithm 5 GetBestAgent from set of agents A for capabilities *capList*
input : Agent set A, list of capabilities *capList*
output: a_{id} such that a_{id} possess the capabilities in *capList* with higher scores

```
1: bestAgent ← nil
2: Ac ← A
3: while not capList.empty() do
4:    r ← capList.removeFirst()
5:    numCandidates ← Ac.count()
6:    i ← 0
7:    while i < numCandidates do
8:       a ← Ac.getFirst()
9:       if not a.possesses(r) then
10:         Ac.remove(a)
11:      end if
12:      i ← i + 1
13:   end while
14: end while
15: score ← −1
16: while not Ac.empty() do
17:   a2 ← Ac.first()
18:   score2 ← id2.getScore(capList)
19:   if  score < score2 then
20:      bestAgent ← a2
21:      score ← score2
22:   else
23:      if  score = score2 and bestAgent.id() > a2.id() then
24:         bestAgent ← a2
25:      end if
26:   end if
27: end while
28: return bestAgent
```

The agent is removed from the set if it possesses the capability (line 9 and 10), this runs in O(1). The code (lines 8 to 12) is part of the while loop in line 7 that iterates over the set of candidate agents, therefore it runs in O(n). The while loop in line 16 also iterates over the candidate agents in O(n). In this loop the score for the required capabilities is calculated and the if-else statements determines the highest (line 17 - 26) which all runs in O(1), making this loop run in O(n). Therefore, the main loop (line 3) runs in O(n^2), thus

the algorithm runs in $O(n^2)$.

4.4.2 Utility Function and Assignment Policies

In DTAACS-OK, the allocation algorithm in the Distributed Task Allocation Component requires other information besides the Organization Knowledge when it generates the assignment set. This extra information is determined by the *Utility Function* used to fulfill the optimization objectives. For example, if the goal is to keep a balanced work load among agents, it would include information about the number of tasks assigned to each agent. This extra information is application dependant. A generic definition is borrowed from [28] where the utility function definition consists of an arithmetic function involving two parameters: expected *quality* of task execution, and expected resource *cost*. The expected quality can be derived from the information about the agent's capabilities stored in the organization knowledge that the Distributed Task Allocation Component receives from the Distributed Organization Knowledge. For the expected resource cost, two sets are defined in DTAACS-OK: the *Assignment Set*, and *Coalition Set*. These two sets are not the only ones than can be considered and a generalization of these entities is presented here as *Utility Criteria Entities*

4.4.3 Utility Criteria Entities

Definition 4.2. *Criteria Entity* is data that needs to be stored to help evaluate a condition or criterion that helps to determine a possible agent-task matching in order to fulfill the optimization objectives.

The criteria entities depend on the optimization objectives like balance load among agents, minimize distance traveled by robot agents, minimize communication cost, etc., which are directly related to the application domain and the user's requirements. DTAACS-OK addresses the single-task type of applications, therefore the allocation algorithm requires the information about the current assignments and coalitions formed. We define these two

entities as follow:

Definition 4.3. The *Assignment Set* is a pair $< agent, task >$ composed by the agent and the task assigned to it.

Definition 4.4. The *Coalition Set* stores the information about the active coalitions in the organization and the agents and task related to the coalition.

Assignment Policies. Assignment policies can be complex and a research topic by itself. In this research assignment policies are simple criteria like (a) agents can be assigned only one task at the time, (b) the closest agent to the destination point gets the assignment, (c) the agent with more successes on executing a particular task type has priority in getting the next task of the same kind, etc.

4.5 Agent's Local Information Component

The Agent's Local Information Component stores the current agent's information. The Agent's Local Information Component interacts with the Distributed Transaction Component. In the case of robot agents, the agent's information is updated by the different sensors the robot possesses. The robot's local information may include physical characteristics of the robot, and its geographical location. Table 4.6 list some agent's information examples.

4.6 Summary

In this chapter, an integrated framework to solve the task allocation problem in cooperative multi-gent systems was presented. The solution is novel in that exploits the information sharing that happens in a cooperative system, specially when near optimal solutions are required. The framework is called DTAACS-OK for **D**istributed **T**ask **A**llocation in **A**daptive **C**omputational **S**ystems based on **O**rganization **K**nowledge. The three main components are Distributed Transaction Component, Distributed Organization Knowledge, and Distributed Task Allocation Component. It also includes the Agent's Current Information

Name	Description
Task Load	When an optimization goals includes work load balance, keeping information of how many tasks the agents is assigned to is requiered
Battery Life	This parameter can be considered when assigning a robot a task that requiters traveling
Camera Type	In heterogeneous systems, robots may posses cameras with different resolutions and zoom ranges
GPS	The precision of the type of GPS possess by a robot can determine who provide the location service (task) in a multi-robot system
Location	The physical location of a robot can determine who to assign a task when minimizing the traveling cost is desirable

Table 4.6: *Agent's Information Examples*

Module that provides the algorithms the agent's information to consider when finding the most suitable agent for the task to be allocated. The distributed allocation of task is computed in each agent by running the same allocation algorithms that require the same input, that is, it required that the organization knowledge that receive as input, it must be in the same state. Therefore, an important task in the framework is to keep the distributed knowledge consistent and provide a mechanism to ensure *one-copy serialiabilty* property. The Distributed Transaction Component is a key part of DTAACS-OK since it executes an atomic commit protocol under a Replica Control Protocol to provide *one-copy serialiabilty* property to the Distributed Organization Knowledge. It consist of a Transaction Generator that fetches the latest agent information when an event like TASK_ACHIEVED occurs and creates a new transaction that is passed to the Transaction Manager.

Chapter 5

Coalitions in DTAACS-OK

Coalition formation in Multi-Agent Systems, particularly in OMAS, is a problem that should not be overlooked. The primary reasons to address coalition formation in MAS include: (1) a single agent may be unable to carry out a task independently, (2) the efficiency to execute a task can be improved by assigning the task to multiple agents, and (3) robustness can result from assigning a task to agents with similar resources. In DTAACS-OK, I tackle a coalition formation problem that considers a grouping of multiple tasks (*complex tasks*) as a single observable entity, which is required by the user in certain applications. For example, in a reconnaissance application in which a human agent participates remotely as part of the robot team, the human agent may require the tasks "AreaOne-Reconnaissance" and "AreaOne-Guard" to be grouped as a single observable task. In general, a user may require feedback from a group of tasks as a single entity, and not from each individual task in the group.

In the following sections, reasons why coalition formation is addressed in this research and a description of a scenario to illustrate the coalition formation problem are presenterd. A taxonomy of the tasks types addressed in this research, the formal definition of the coalition formation problem, and the algorithms to form the coalitions are also presented.

5.1 Motivation and Problem Illustration

The benefits of assigning a task to a coalition of agents are indicated in several papers, such as those presented by Tang and Parker in [49] and Shehory and Kraus in [45]. Solutions discussed in these publications address most challenges found in a multi-agent system in which a task to be achieved requires more than one agent. Shehory and Kraus present [46] an algorithm that can generate coalitions in which agents may belong to one or multiple coalitions at the same time, but the cost of communication to form, keep, and terminate the coalitions is high. Also, the assumption that agent resources can be shared among coalition members is not applicable to some OMAS, such as systems in which the agents are robots. Vig and Adams offer [52] an improvement to Shehory and Kraus' algorithm that eliminates this assumption, but the communication cost remains high and the calculation of the number of agents in a coalition, a key input to the algorithms, is not clearly defined. To the best of my knowledge, none of the solutions proposed in the literature address the need to monitor the status of a group of tasks.

Three issues that motivate further research of coalition formation in OMAS are identified: (1) high communication cost to form, maintain, and terminate a coalition, (2) lack of clarity on how to determine the minimum number of agents in a coalition, which is a key algorithm parameter, and (3) user requirements to get feedback specifically from complex tasks.

To illustrate the problem, a description of the *Site Clearing Problem* is given in Section 5.1.1. Various task categories that can be part of the input of the algorithms in this research are introduced in Section 5.1.2. The categories are defined based on the location of required resources specified by the task and whether the tasks are specified as a logical point of observation.

5.1.1 The Site Clearing Problem

The *Site Clearing Problem* (SCP) is presented here to illustrate the need for forming coalitions in order to achieve a common goal. The problem of clearing a site in which different

objects have been seeded has been described by Tang and Parker in [49]. One reason for borrowing the SCP is its similarity to the problem addressed in HuRT-IED, a research project in the MACR Lab, and also, as mentioned by Tang and Parker in [49], this application has been identified by NASA as an key prerequisite for human missions to Mars. The HuRT-IED scenario is described in detail in Chapter 6 in this dissertation.

Site Clearing Problem Description (SCP). This problem addresses a scenario in which there is a predefined area A, and objects of different sizes and weights are dispersed in unknown locations within the area. The goal in this problem is to find the objects and remove them from the area A. For generalization purposes, the optimization objective is not specified, but it can be the minimization of (1) the execution time, (2) the distance traveled by the robots, or (3) the communication among robots, or other optimization objectives. The following constraints are set in order to make this problem more suitable to this research.

1. The area is divided into subareas for which the user requires updates on the status of the clearing task.

2. The objects to be removed may require more than one robot. The objects are classified as Obj-A, Obj-B, and Obj-C, as specified in Table 5.1.

3. The team is composed of heterogeneous robots (Table 5.2), but all robots possess a scanner to detect objects.

4. The removal of objects that require more than one agent have precedence over tasks that require only one robot as well as search area tasks. A task tree for the SCP is depicted in Figure 5.1

Because the tasks specify what capabilities are required, agents that possess those capabilities and the possible grouping of those agents must be identified. As suggested by Shehory and Kraus in [45], in regards to communication and computational cost, it is better to find the smaller coalition size possible. As mentioned, SCP is similar to the problem of localizing

and defusing Improvised Explosive Devises (IEDs) addressed in the military domain, where minimizing the communication cost is important because it minimized exposure of sensitive information, and possible localization of troops by the enemy.

In order to comply with constraint 1 in this example, feedback to the human agent is required about the area A being totally clear, and not about an agent completing the task of removing one object, or possible multiple objects located in a section of the area A.

From the task tree depicted in Figure 5.1, relevant task characteristics for this research are identified, and discussed in Section 5.1.2 as a task taxonomy.

Table 5.1: *Clearing Site Objects*

Object Type	Weight	Volume
Obj-A	10 lb	0.5 cu ft
Obj-B	15 lb	0.75 cu ft
Obj-C	25 lb	1.25 cu ft

Table 5.2: *Clearing Site Agents*

Agent	Push Capability	Carry Capability	Scanner
a01	10 lb	-	√
a02	10 lb	0.5 cu ft	√
a03	15 lb	1.5 cu ft	√

5.1.2 Tasks Taxonomy

In this section, tasks are classified based on characteristics that are relevant to coalition formation algorithms. Other task characteristics that were not considered include whether a task has a specific *begin* and *end* states (discrete task), whether a task does not have a *finish state* (continuous task), and whether a task has a *start* and *end* time (scheduling). The taxonomy presented here is derived from the definitions of *Tight Coordination-Simple Task* and *Tight Coordination-Complex Task* (Definitions 3.2 and 3.3, respectively), and Definition 5.1, which defines an observable task.

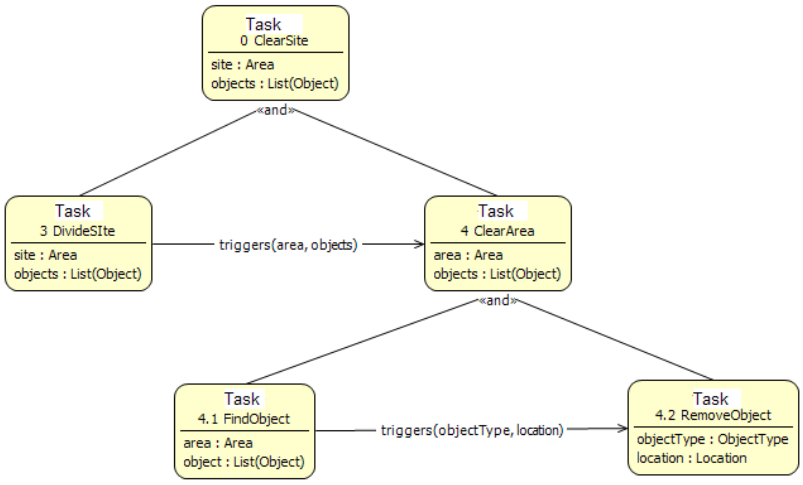

Figure 5.1: *Site Clearing Task Diagram*

Definition 5.1. Observable Task

An *observable tasks* is a task that, regardless of its type, the user requires feedback on its status.

The task taxonomy is presented in three groups (Table 5.4) determined by the criteria in Table 5.3.

Table 5.3: *Group Criteria*

	Criterion
Group 1	Tasks considered in previous published coalition formation algorithms
Group 2	Tasks not considered in previous coalition formation algorithms
Group 3	Tasks not relevant to the systems considered in this research (see Table 5.4)

The type of tasks in Groups 1 and 2 are addressed in this research. In addition to considering the type of task in Group 2 in DTAACS-OK, significant differences exist between the solution proposed in this research and those presented in market-based approaches. These differences are derived from the fact that DTAACS-OK uses a distributed organization knowledge when deciding about coalition formation. The differences are discussed in more detail in Section 5.2.

Table 5.4: *Task Taxonomy for Coalition Formation*

	Type of Task
Group 1	**OLC-ST**: Observable Lightly Coordinated Simple Task. **OTC-ST**: Observable Tightly Coordinated Simple Task.
Group 2	**OLC-CT**: Observable Lightly Coordinated Complex Task. **OTC-CT**: Observable Tightly Coordinated Complex Task.
Group 3	**NOLC-ST**: Not-Observable Lightly Coordinated Simple Task. **NOLC-CT**: Not-Observable Lightly Coordinated Complex Task. **NOTC-ST**: Not-Observable Tightly Coordinated Simple Task. **N-TC-CT**: Not-Observable Tightly Coordinated Complex Task.

5.2 Coalitions in DTAACS-OK

In Chapter 3, a Tight Coordination-Simple Task (TC-ST) was defined as a task that cannot be decomposed and requires resources that reside in more than one agent. Coalition formation solutions proposed in the literature consider this type of task and, if a single agent coalition is considered, include Light Coordination-Simple Task (LC-ST) tasks as well. In this research Tight Coordination-Complex Task (TC-CT), described in Definition 3.3, are also included. The problem of forming coalitions stated in Section 5.3 is tackled in DTAACS-OK by answering the following questions:

Question 1: When a coalition is needed?

Question 2: How is the size of a coalition computed?

Question 3: Which coalition to use?

Question 4: Who requests the coalition formation?

Question 5: Who terminates the coalition?

The candidate coalition formation algorithm in this research is inspired by algorithms presented by Shehory and Kraus in [46] and improved on by Lovekesh and Adams in [52].

However, the way answers to the questions are generated is different because agents in DTAACS-OK use identical distributed organization knowledge in order to make decisions.

When a coalition is needed? In DTAACS-OK a coalition formation is determined by: (1) the capabilities required by the tasks or (2) a complex task specified as an observable task. Information regarding the capabilities required by the tasks is considered in published algorithms, and tasks specified as an observable task is added in DTAACS-OK in order to address the user need to gain feedback from a specific task.

How is the size of a coalition computed? In the algorithms proposed in [52] and [45], the suggestion is made that a minimum number of agents (k) should be used and that calculation of the number k could be based on the requirement that all tasks must be executed by the same number of agents. However, if such requirement is not specified in the problem, it is suggested to use the minimum possible value for k. In algorithms presented in this research, the requirement of the same number of agents for all tasks is not expected. Candidate coalitions for a task are generated beginning with one agent and incrementing the number of agents by one until a coalition that can execute the task is found.

Which coalition to use? In DTAACS-OK, this question is not answered when coalitions are formed. The coalition that is allocated a task is determined by the allocation algorithm (Chapter 4) and depends on scalar value calculated by the cost function and optimization objective.

Who requests the coalition formation? and *Who terminates the coalition?* In DTAACS-OK, no specific communication to request the formation or termination of a coalition is needed. The agents in the organization learn about being part of a coalition by executing the candidate coalition formation algorithms, usually after processing a TASK_ACHIEVE, TASK_FAILED, TASK_TRIGGERED, or TASK_NEGATIVE_TRIGGERED messages (Chapter 4). The agents know they are not part of the coalition after achieving or failing the task and they consequently send the appropriate message to the rest of the agents in the organization.

Before presenting algorithms to generate coalitions in DTAACS-OK, the problem formulation in order to form the candidate coalitions is posed in Section 5.3.

5.3 Problem Statement

In this section, the problem to generate the *candidate coalitions* for each task is formulated. Finding the best coalition from all possible coalitions for a task is a combinatorial problem that is NP-Complete. As in [45], the goal is to obtain an algorithm that runs in polynomial time by setting a maximum number of agents that can be in a coalition. The *Candidate Coalition Formation* problem definition is as follows:

> Given a set of Task (T) where each task t_i ε T requires some capabilities to be achieved (C_{t_i}), and having a set of Agents (A) where each agent a_i ε A possesses some capabilities (C_{a_i}), the problem is to generate, for each task $\boldsymbol{t}$, the set or sets of agents that can achieve $\boldsymbol{t}$.

The *candidate coalition* is obtained by restricting the *coalition* definition presented in Definition 3.5. The restriction is defined by the *achieves* function specified below. The *achieves* function uses Definition 3.4 (*required capability set*) and Definition 5.2 (*coalition capability set*).

Definition 5.2. Coalition Capability Set C_τ

C_τ is the set that includes all capabilities possessed by the agents in the coalition τ and is given by: $C_\tau = \bigcup_{a \varepsilon \tau} C_a$

Definition 5.3. Achieves Function

$$achieves(\tau, t) = \begin{cases} false & \text{if } C_t \supset C_\tau \\ true & \text{if } C_t \subseteq C_\tau \end{cases}$$

where C_t is the required capability set for task t, and C_τ is the coalition capability set.

Definition 5.4. Candidate Coalition

A *candidate coalition* is defined as the tuple $< \tau, t >$; where τ is the set of agents such that achieve(τ, t) is *true*.

The algorithms presented in Section 5.4 solve the *candidate coalition* problem and generate as output the set of coalitions for each task that is ready for allocation.

5.4 Coalition Algorithms

Before introducing the algorithms specified in this proposal, the definition for *task capability matrix* (TCM) and *conforms* function are presented. The task capability matrix definition is borrowed from Vig and Adams [52] and presented here in a formal way. By creating the TCM and verifying the capabilities each agent possesses against the matrix, the problem of agents not being able to transfer capabilities, such as in multi-robot systems, is addressed.

Definition 5.5. Task Capability Matrix

The matrix M_t is a constraint representation for the capabilities required by task $\boldsymbol{t}$. Each entry in M_t represents a pair of capabilities required by t, and the constraint is:

$$M_{ij} = \begin{cases} 0 & \text{if } c_i \text{ and } c_j \text{ must reside in different agents} \\ 1 & \text{if } c_i \text{ and } c_j \text{ must reside in the same agent} \\ -1 & \text{if } c_i \text{ and } c_j \text{ are not restricted by their location} \end{cases}$$

The *conforms* function determines if an agent possesses a pair of capabilities required by task t that satisfies the constraint in M_t.

Definition 5.6. Conforms Function

$$conforms(a, Mt) = \begin{cases} 0 & \text{if } \boldsymbol{a} \text{ does not possess any capability pair that satisfies some entry in } Mt \\ 1 & \text{if } \boldsymbol{a} \text{ does possess at least one capability pair that satisfies some entry in } Mt \end{cases}$$

5.4.1 A General Overview of Candidate Coalitions Generation

A brief description of the algorithms that generate the candidate coalitions is as follows.

1. Take the Agents A and Task T sets as input.

2. For each task t ε T, generate the matrix M_t that determines capabilities the task requires in order to reside in the same agent.

3. Generate the set cA_{t_i} of candidate agents for each task by filtering the agents A in order to keep only the ones that conform to the TCM M_t.

4. Considereing t and cA_{t_i}, generate the smallest coalition or coalitions for each task in the set T.

5. Return the set of coalition sets.

In this research, multiple agent tasks have priority over single agent tasks and in the case when an agent is assigned to a single agent task, the agent's scheduler is responsible for rescheduling the single agent task currently executed so that the agent begins execution of the task allocated to the coalition it part of. Agents can belong to multiple candidate coalitions, but can be actively part of just one coalition.

5.4.2 GetBestCoalition Algorithm

The GetBestCoalition algorithm (Algorithm 6) selects the most suitable coalition from the set of candidate coalitions for the task t_i. The algorithm first initializes the suitable coalition to null (line 1), then calls algorithm MainCoalitionFormation (algorithm 7) to generate the set of candidate coalitions (line 2). Once the candidate coalitions are generated, the algorithm loops (line 3) and takes each candidate coalition (line 4) and calculates the suitability value of each coalition (line 5). The value of the current best coalition and the next coalition are compared (line 5) and if the candidate coalition is more suitable than the current best, the best coalition is replaces by the candidate coalition (line 6). When all the candidate coalitions are evaluated, the loop ends (line 8) and the most suitable coalition for the task is returned (line 9).

GetBestCoalition Algorithm Complexity Analysis

Time Complexity: $O(n^2)$

Algorithm 6 GetBestCoalition for A, t_i
input : Organization Agents (A) and task t_i
output: The coalition c for task t_i

```
1: coalition ← null
2: coalitions ← MainCoalitionFormation(A, t_i)
3: while coalitions is not ∅ do
4:    coal ← coalitions.removeFirst()
5:    if value(coal) > value(coalition) then
6:       coalition ← coal
7:    end if
8: end while
9: return coalition
```

Reasoning: The algorithm begins by initializing the coalition to the null set which runs in O(1) , and the set of coalitions for a given task (line 1 and 2) runs in $O(n^2)$. Then, the *while* loop iterates over the set of coalitions to determine which one is the best for the task (line 3, 4, 5, 6 and 7) which runs in O(n) time. Thus, the algorithm *GetBestCoalition* runs in $O(n^2)$.

5.4.3 MainCoalitionFormation Algorithm

The MainCoalitionFormation algorithm (Algorithm 7) generates candidate coalitions for the task ready to be allocated. The algorithm generates the task capability matrix M_t (line 2) in order to determine the resources task $\boldsymbol{t}$ requires to reside in the same agent. Agents in A are filtered in order to retain only those that *conform* with matrix M_t (line 3). Line 4 and 5 set the initial minimum and maximum number of agents in a coalition. Next, the set of candidate coalitions for task $\boldsymbol{t}$ is initialized (line 6). The while loop (line 7) controls the search of a candidate coalition with the minimum number of agents possible (line8). The loop iterates until a coalition is found or the maximum number of agents is reached. When at least one coalition is found for a task, that coalition is added to the list of candidate coalitions (line 11) in order to be returned (line 13). (The assumption is made in this research that at least one coalition exists for each task.)

MainCoalitionFormation Algorithm Complexity Analysis

Algorithm 7 MainCoalitionFormation for a_i
input : Organization Agents (A) and ready to assign task t_i
output: Set of Coalitions for task t_i

1: $coalitions \leftarrow \emptyset$
2: $taskAM \leftarrow$ createTaskAM(t_i)
3: $candidateAgents \leftarrow$ filterCandidateAgents(A, t_i, taskAM) -See Algorithm 6
4: $numAgents \leftarrow candidateAgents.numAgents()$
5: $k \leftarrow 1$
6: $newCoalitionsForTask \leftarrow \emptyset$
7: **while** ($newCoalitionsForTask$ is $\emptyset$) and ($k \leq numAgents$) **do**
8: $\quad newCoalitionsForTask \leftarrow$ coalitionsForTask(t_i, $candidateAgents$, k) -See Algorithm 6
9: $\quad k \leftarrow$ k + 1
10: **end while**
11: $coalitions \leftarrow coalitions \bigcup newCoalitionsForTask$
12: return $coalitions$

Time Complexity: $O(n^2)$

Reasoning: The algorithm initializes the coalition set to the empty set (line 1) which runs in constant time, O(1). The *task allocation matrix* is generated (line 2) which runs in $O(n^2)$. The candidate agents are filtered (line 3) which runs in O(nlgn). Assigning *numAgents*, k, and *newCoalitionsForTask* to the number of candidate agents, 1 and the empty set, (line 4, 5 and 6) respectively, all run in O(1). Finding the coalitions for a given task (line 8) runs in O(n), so looping until $k >$ *numAgents* (line 7) runs in $O(n^2)$. Finally, creating the union of two sets (line 11) runs in O(n). Therefore, the algorithm *MainCoalitionFormation* runs in $O(n^2)$.

5.4.4 CoalitionsForTask Algorithm

The algorithm CoalitionsForTask (Algorithm 8) has as input the task $\boldsymbol{t}$ that requires a coalition, the candidate agents that possess some of the capabilities required by task $\boldsymbol{t}$, and the number of agents in a coalition. This algorithm generates all candidate coalitions for task $\boldsymbol{t}$ with up to $\boldsymbol{k}$ agents (line 1). Once possible coalitions are formed, the algorithm verifies them (line 3 and 4) in order to select the ones which agents by contributing their capabilities fulfill all capabilities required by task $\boldsymbol{t}$ (line 5) ; those coalitions are added (line 6)

to the set of candidate coalitions to be returned (line 9). **Algorithm CoalitionsForTask**

Algorithm 8 CoalitionsForTask t up to k agents
input : Task t, Candidate Agents (cAgents), max k
output: Set of coalitions for task t

```
1: possibleCoalitions ← combinationsOf(candidateAgents, k)
2: cCoalitions ← ∅
3: while possibleCoalitions not ∅ do
4:     c ← possibleCoalitions.getFirst()
5:     if coalitionForTask(c, t) then
6:         cCoalitions.add(c)
7:     end if
8: end while
9: return cCoalitions
```

Complexity Analysis

Time Complexity: O(n)

Reasoning: The algorithm creates a set of possible coalitions (line 1) which runs in O(nlogn). All other assignments (line 2 and 4) similarly run in O(1). Checking that all capabilities contained within the agents of each coalition are sufficient for the task (line 5) also run in O(1). Adding to the set of candidate coalitions run in O(1). Thus, looping through the possible coalitions, removing the first element each run, runs in O(n); where n is dependent on the number of possible coalitions. Thus, the *CoalitionsForTask* algorithm runs in O(n).

5.4.5 FilterCandidateAgents Algorithm

The algorithm FilterCandidateAgents (Algorithm 9) determines the candidate agents for a task based on capabilities required by the tasks; taking into consideration the restriction that some capabilities must be in the same agent. As stated by Vig and Adams in [52], the capabilities that a robot possesses cannot be easily transferred to another robot while working on a mission. To address this problem, Vig and Adams proposed in [51] to form a capability matrix that represents the constraints for a pair of capabilities to reside in the same robot. Therefore, a matrix for each task is later represented as a Constraint Satisfaction

Problem (CSP) (line 1). The CSP can be solved using a backtracking, forward checking, or maintaining arc consistency algorithm [6] (line 2). The CSP helps determine if an agent can be part of a candidate coalition for task ***t***. (line 3) **Algorithm FilterCandidateAgents**

Algorithm 9 FilterCandidateAgents(Set, Matrix, task)
input : *candidateAgents*, task t, and *taskAM*
output: *candidateAgents* filtered based on the *taskAM*

1: Formulate the *taskAM* as a CSP
2: Solve the CSP formulated to filter the *candidateAgents*
3: return *candidateAgents*

Complexity Analysis

Time Complexity: O(nlgn)

Reasoning: The algorithm creates a CSP out of the Task Allocation Matrix provided (line 1) which runs in O(1). To solve the CSP in order to filter the candidate agents (line 2) runs in $O(nd^n)$ where n is the number of nodes and d^n is their cost of computing. In DTAACS-OK D is a binary set making $|d| = 2$, thus making the algorithm run in O(nlgn).

5.5 Summary

In this chapter, the *Candidate Coalition Formation* problem addressed in DTAACS-OK was presented. In Section 5.1, the reason to form coalitions in an OMAS was highlighted by describing the Clearing Site Problem. The type of task handled by coalition formation algorithms in DTAACS-OK were grouped based on task characteristics of (1) where the task required-resources reside in the robots, and (2) whether the user defined a task as an observable point. The formulation of the problem was posed in Section 5.2, and three definitions were introduced: (1) Coalition Capability Set (Definition 5.2), (2) Achieves Function (Definition 5.3), and (3) Candidate Coalition (Definition 5.4). The pseudo code for the algorithms, their description, proof of correctness, and complexity analysis were presented in Section 5.4.

Chapter 6

DTAACS-OK Empirical Evaluations

In this chapter, preliminary results of evaluations of DTAACS-OK in a simulated application are presented. As stated before, the framework proposed in this research aims to provide a distributed, adaptive, an efficient solution to the task allocation problem in MAS. In particular efficient in regards to communication.

6.1 DTAACS-OK for HuRT-IED

6.1.1 Motivation

A common situation in the military battle field is to explore and scan areas for detection and removal of Improvised Explosive Devices (IEDs). This type of mission increases the security and the safety of the convoy before it occupies the area. In this type of scenario, some of the important factors to consider are communication, robustness and time to complete the mission. Even though communication in this scenario may not be as limited as in other domains like underwater mine recovery, it is highly desirable to reduce the communication among the robots in order to reduce the information that may be exposed for interception. Robustness in most domains is required, and this is no exception. If a malfunction or robot loss occurs, the team needs to recover and adjust promptly. As in search and rescue applications, completing the mission of scanning and defusing IEDs in the minimum time is key to avoid possible exposure to the enemy.

The framework was tested in this scenario by integrating DTAACS-OK into the HuRT-IED application. The HuRT-IED application was implemented as part of the Human Robot Team (HuRT) research conducted at the Multi-Agent and Cooperative Reasoning Laboratory at Kansas State University [34]. The scenario is defined by an area to be explored, a set of robots as field agents, a number of suspicious objects (SOs), and a human agent. The field agents have the capabilities to scan, identify, and dispose of the IEDs. The SOs are randomly placed in the area, and for some of them, the field agents require the help of the human agent to determined if an SO is an IED or not.

6.1.2 Mission and Task specification

In this particular mission the area to be scanned is a cross road as shown in Figure 6.1. The Task Tree for the mission is depicted in Figure 6.2. The main task is to clear the given area from IEDs. We decomposed this main task into sub-tasks as follow:
Task- 0 Clear Area. This is the first task that exists when the system starts.

Task- 1: Load Scenario. This task is triggered by the Clear Area task. It loads the area to be cleared and the expected number of robots. The number of SOs is unknown.

Task- 2: Search Area. This task is triggered by the Load Scenario task and it is divided into the following subtasks:

Task-2.1: Divide Area. This task takes the initial area to be scanned and, depending on the team's configuration (number of agents, location of agents, etc), triggers the Scan Area task passing a subarea to be scanned. This task helps test the framework in regards to team collaboration.

Task-2.2: Scan Area. This task is one of the three main activities of the Clear Area task. When pursuing this task, if a SO is found, the Identify Object task is triggered with the SO location.

Task- 3: Identify Object. This task is triggered by the Scan Area task when an SO has been found. To provide the flexibility for other agents to try to identify the SO as an IED (rather than the one that found the object), the task is divided as follows:

Task-3.1: Robot Identification. This task is pursued first in the identification process. If the agent trying to identify the SO cannot determine if it is an IED or not, it triggers the Human Identification task, providing as much information as possible (picture and sensor levels).

Task-3.2: Human Identification. This task is triggered by the agent executing the Robot Identification task when it fails to determine if the SO is in fact an IED or not. The Human Agent is prompted and makes a decision based on the information provided by the field agent.

Task- 3: Defuse IED. This task is triggered by the Identify Object task when an SO has been positively identified as an IED, either by the robot or the human agent.

The mission is achieved when the search area is scanned, all the SOs found have been identified, and all IEDs have been defused.

6.1.3 General Scenario Description

The framework was evaluated against the following scenario. In this particular mission, the area (A) to be scanned for IEDs was an intersection of two roads. Also, there were five agents initially positioned at a predefined location at the entrance of one of the roads as depicted in Figure 6.1. When the mission starts, and after the agents have registered with each other, the area A is divided by the agent that possesses the *DivideArea* capability. That agent divides the area into subareas of similar size based on the current number of agents in the team. Once the subareas are defined, the tasks to scan the subareas are assigned to the agents in a nondeterministic way since at the start all the agents are located basically in the same position. Each agent starts scanning the assigned area following the greedy coverage

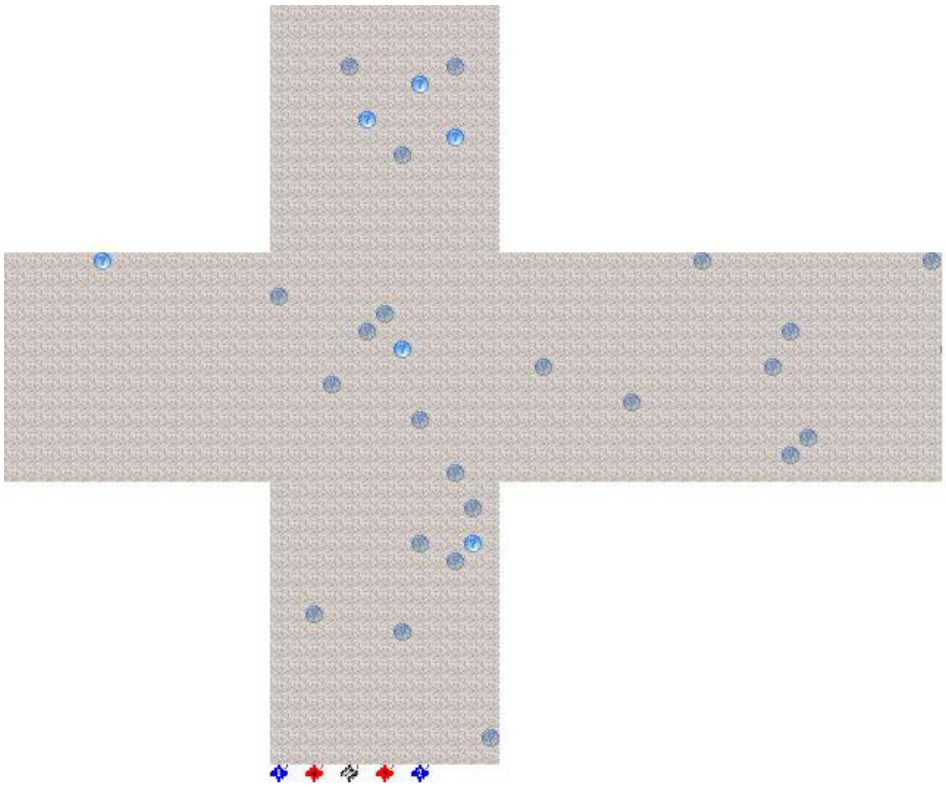

Figure 6.1: *HuRT-IED Scenario*

area algorithm that selects the closest unvisited location and generates a direct path to the location, avoiding any obstacles it finds in its path. If there is more than one location with the same distance, the agent randomly selects one. The SOs were randomly distributed all over the roads. For each treatment, from one to thirty SOs were seeded in increments of one. To achieve the tasks in the mission, the team of robots needed to be able to scan, identify, and defuse the IEDs. The three types of agents used in this framework evaluation are shown in Table 6.1. There were a total of five agents in each mission: Two Scanners agents (S-1 and S-2) were capable only of finding suspicious objects, and thus could only trigger Identify Object tasks. One Identifier agent, (SI-1), was able only to detect, and possibly identify, but not defuse IEDs; therefore, this agent could trigger Defuse IED and Identify Object tasks and, depending on the object found, Human Identify tasks. The last two field agents were Defusers (SID-1 and SID-2), which were capable of detecting, identifing and defusing IEDs.

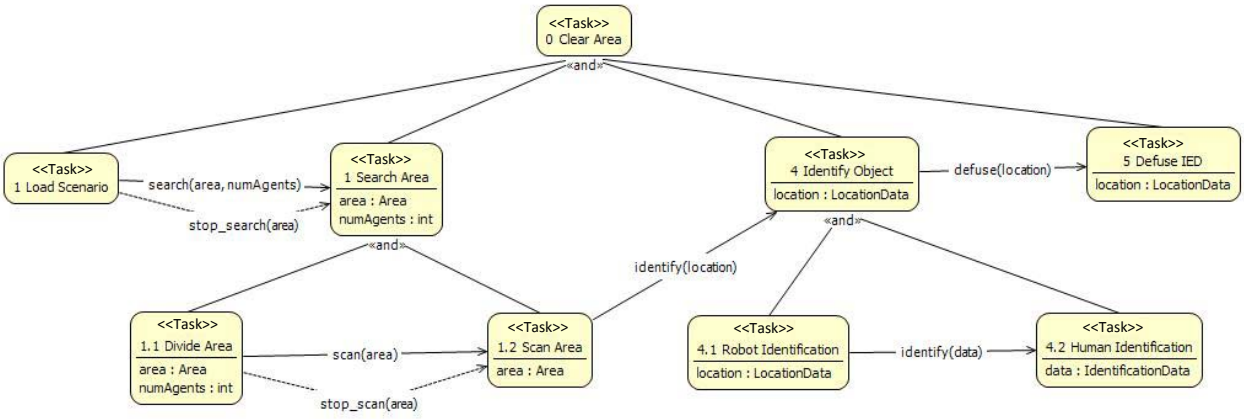

Figure 6.2: *HuRT-IED Tree Mission Representation*

When agents collided, they saw each other as obstacle as well. Agents possessed a simple obstacle avoidance algorithm. Agents could work on one task at the time and their internal

Name	Scan	Identify	Defuse
Scanner (S)	√		
Identifier (SI)	√	√	
Defuser (SID)	√	√	√

Table 6.1: *Agent Types*

task scheduler handled the task priority defined next. Load Scenario and Divide Area tasks were executed during the initialization phase with Load Scenario triggering Divide Area. The other tasks with more than one instance during mission execution were as follows: Defuse had the highest priority, Machine Identification was the second highest, and Scan had the lowest priority. When a agent detected a new task with a higher priority then the one currently executing, it saved the task progress information and swaped the task. The released task was placed in a waiting queue. The waiting task priority was incremented by one each time the agent postponed its execution because another task with higher priority arrived. For the framework evaluation, the independent variables are listed in Table 6.2 while the dependent variables are listed in Table 6.3.

6.1.3.1 Scenario Evaluation based on the SOs types

As mentioned in Section 6.1.3, there are different types of SOs: (1) Garbage (*G-1*), which are identified by robot agents, (2) Garbage (*G-2*), which are identified only by human agents, (3) *IED-1* identified by robot agents, and (4) *IED-2* identifiable only by human agents. Depending on these type of SOs, different types of task are generated in the mission, hence, different number of messages are generated. Three examples are described below to illustrate how the types of SOs affect the number of messages generated in a mission.

1. The case when the area A is clear of IEDs, that is, all SOs are garbage of the type *G-1* and no *HumanIdeification*, or *Defuse* tasks are generated, therefore, less messages will be generated.

2. The case when all IEDs in the area A are identifiable only by human agents. In this case *HumanIdeification*, and *Defuse* tasks are generated, having the most messages types generated in a mission.

3. The case when there is a mix of SOs types, some are G-1, some G-2, some IED-1, and other IED-2. This case represents a most realistic scenario.

The different treatments are described in Section 6.1.4, where the three cases described above are considered.

6.1.4 Particular Scenario Specification

Treatment E1. E1 was the control. The independent variables message-loss and capability-degradation were set to zero. All agents had a full team broadcast communication capability. Data was collected about the number of messages transmitted. This treatment is divided in **E1a**, which is the one when all SOs are garbage and tasks to request the human identification or to defuse IED are not generated. Treatment **E1b** considers the case when all

ID	Name	Description
IVar-1	Messages Loss	Since DTAACS-OK is a knowledge based solution for distributed task allocation, it is important to show how the framework behaves under less than favorable communication conditions
IVar-2	Capability Degradation	One of the main factors that determine if an agent is the most suitable for a task is the state of the set of capabilities it possesses. (Other factors may include assignment policies, or optimization goals such as load balance among agents). It is important to show how the framework performs when the agent's capabilities changes to the point the agent fails the task. Also, to verify that it is not necessary to send a message when a capability changes. The hypothesis here is that when a capability changes, the change may not be severe enough to change the suitability of the agent form most suitable to a lower level, or vice versa. (The last condition may occur when the distance from the current agent's position to a destination determines who gets the assignment).
IVar-3	Number of Suspicious Objects	To demonstrate how the framework performs in relation to scalability, we seeded different numbers of suspicious object to search, identify, and defuse.

Table 6.2: *Independent Variables*

the SOs require the human agent for further identification and all SOs are IEDs, therefore, *HumanIdeification*, and *Defuse* tasks are generated. Treatment **E1c** considers the case when all the SOs are a mix of G-1, G-1, IED-1, and IED-2 with 25% of each in each run. Results over a hundred runs for scenarios seeded with one to thirty SOs are shown in Figure 6.3. The dark blue line represents the average messages sent when there is no communication problems (**E1a**). As expected, the number of messages is linear as discussed in Section 4.2.3.3, where it is shown that the message complexity is in $O(n)$. The result for **E1b** are shown in Figure 6.4, and Figure 6.5 for **E1c**. In both figures the dark blue line represents the number of messages when no communication loss. Both of them are linear as expected.

ID	Name	Description
DVar-1	Communication Cost	The number of messages sent to achieve the mission
DVar-2	Execution Time	The time spent to achieve the mission in terms of simulator turns
DVar-3	Task Re-Allocations	Number of times a task is reallocated due to agent unable to pursue a task because a capability degradation

Table 6.3: *Dependent Variables*

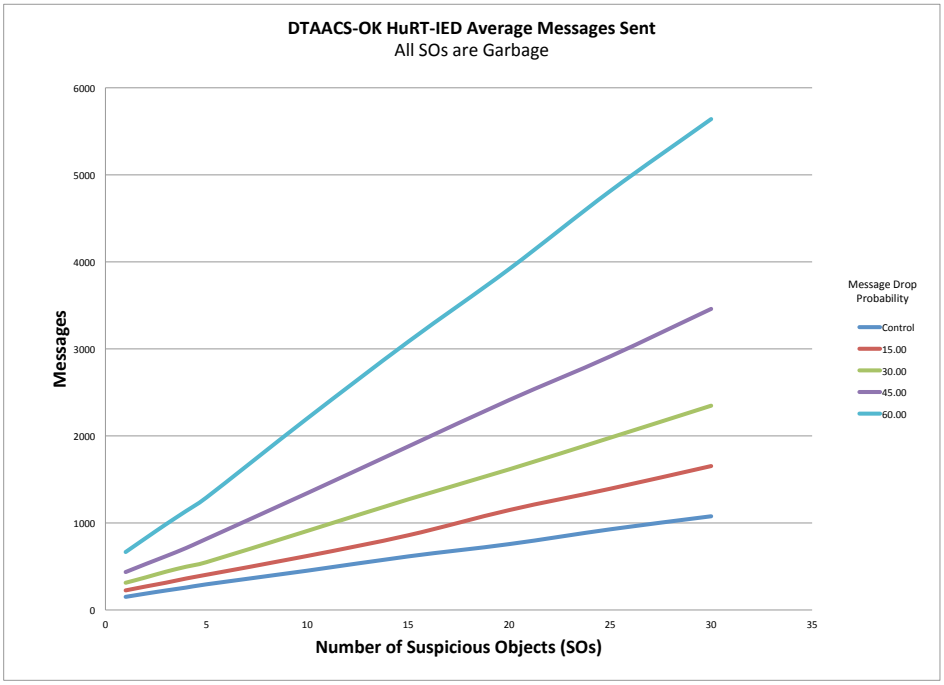

Figure 6.3: *HuRT-IED Average Message Sent all SOs are G*

Treatment E2. For E2 the assumption of perfect communication conditions was relaxed. DTAACS-OK was tested under 0%, 15%, 30%, 45%, and 60% probability of loosing a message with no communication delay and full broadcast capability. There was no capability degradation for this treatment. Again, this treatment is divided to evaluate the cases

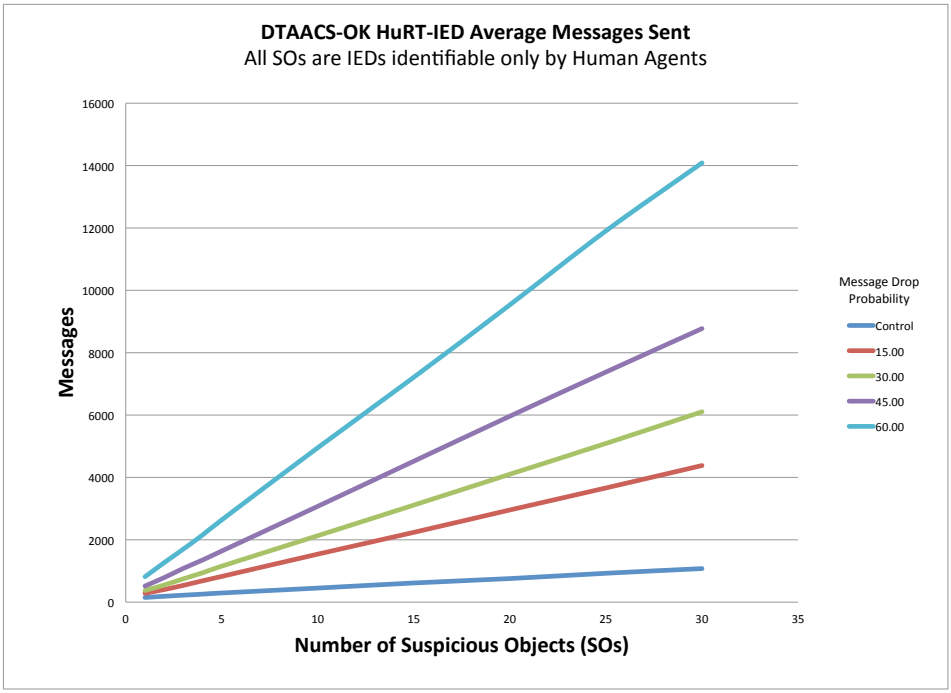

Figure 6.4: *HuRT-IED Average Message Sent all SOs are IEDs identifiable only by Human Agent*

described in Section 6.1.3.1. **E2a** addresses the case when all SOs are garbage and tasks to request the human identification or to defuse IED are not generated. Treatment **E2b** considers the case when all the SOs require the human agent for further identification and all SOs are IEDs,therefore, *HumanIdeification*, and *Defuse* tasks are generated. Treatment **E2c** considers the case when all the SOs are a mix of G-1, G-1, IED-1, and IED-2 with 25%, of each in each run. Figure 6.3 shows the results for average number of messages over a hundred runs for the same condition regarding IEDs as in ***E1a***. The red, green, purple and light blue lines, show the number of messages sent for 15%, 30%, 45%, and 60% probability of loosing a message. The results are linear as expected and the increase in the slope of each treatment reflects the increase in the probability to lose a message, which was also expected. Figure 6.4 shows the results for average number of messages over a hundred runs for the same condition regarding IEDs as in ***E1b***. Again, the red, green, purple and light

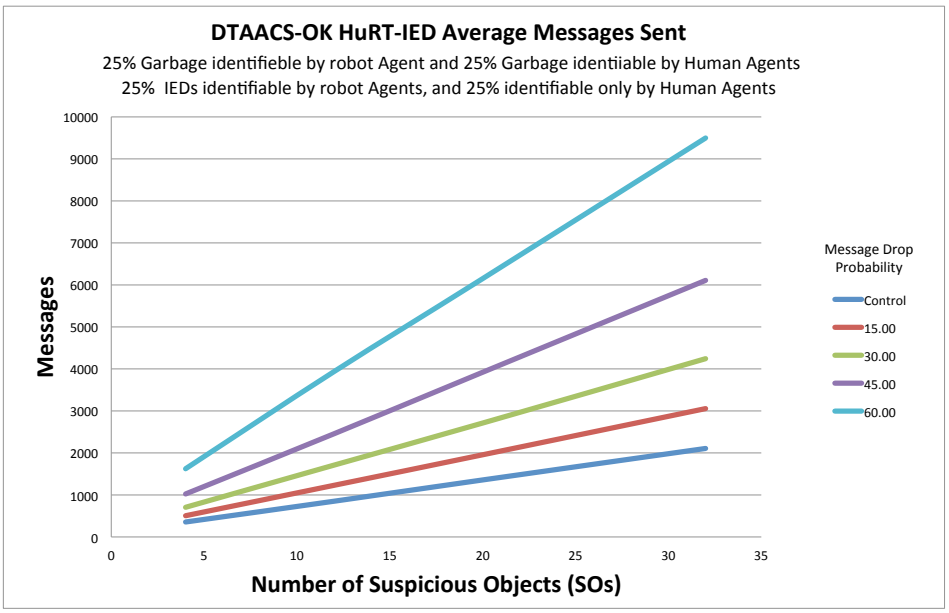

Figure 6.5: *HuRT-IED Average Message Sent all SOs are Mix*

blue lines, show the number of messages sent for 15%, 30%, 45%, and 60% probability of loosing a message. The results are linear as expected and the increase in the slope of each treatment reflects the increase in the probability to lose a message, which was also expected. Compared to the results in Figure 6.3 for ***E2a***, the number of messages are higher and the reason is there are tasks generated because of the seeded SOs that are not present in ***E2a***, this was also as expected. For ***E2c***, Figure 6.5 shows the results for average number of messages over a hundred runs when the SOs are mixed. The results for each treatment are linear. The number of messages in this case fall between the results of ***E1a***and ***E1b*** because they reflect the different types of task generated in the system because of the different types of SOs seeded.

Treatment E3. In the third treatment the framework was stressed by introducing agent capability degradation to the point that agents were not able to execute and achieve their tasks. The number of messages sent for task failure scenario were thus counted. The agents had full broadcast communication, no delays and no message drops, and there was at least

one agent in the organization capable of executing all tasks that remained. For this treatment, when the agent's sensor, camera, and gripper capabilities de-gradated, they did not recover. This treatment was also divided into **E3a** and **E3b** to address the conditions where all SOs were *G-1* and all SOs were *IED-2* respectively. Results for the average number of messages sent due to agents' capability degradation over a hundred runs are shown in Figure 6.6 for case **E3a** when all SOs are garbage identifiable by the robot agents. The dark blue line represents the average messages sent when there is no task failures. The red, green, purple and light blue lines, show the average number of messages sent for 15%, 30%, 45%, and 60% probability of failing a tasks. Failing a task introduces more tasks in the system that require extra messages to be sent, which can be seen as an increment of the factor that multiplies the message function discussed in Section 4.2.3.3. The results are linear as expected and the increase in the slope of each treatment reflects the increase in the probability to fail a task, which was also expected. Similarly, Figure 6.7 depicts the results for **E3b** when all SOs are IED identifiable only by the human agent.

Treatment E4. In treatment E4 a more realistic simulation was used to test DTAACS-OK. E4 is combination of **E2b** and **E3b** where communication is not perfect and agents' capabilities may fail in some degree and cause reallocation of tasks, and all SOs are IED that require the human agent intervention. Figure 6.8 shows the results of a hundred runs for communication conditions where there was 15% probability to drop a message and 15% probability that an agent fail to achieve a task, and therefore a reallocation occurred. In both, **E2b** and **E3b** result, it is shown that the average number of messages sent are linear and bound by $O(n)$, for these reasons, it was expected that also the results for E4 were linear and Figure 6.8 shows it.

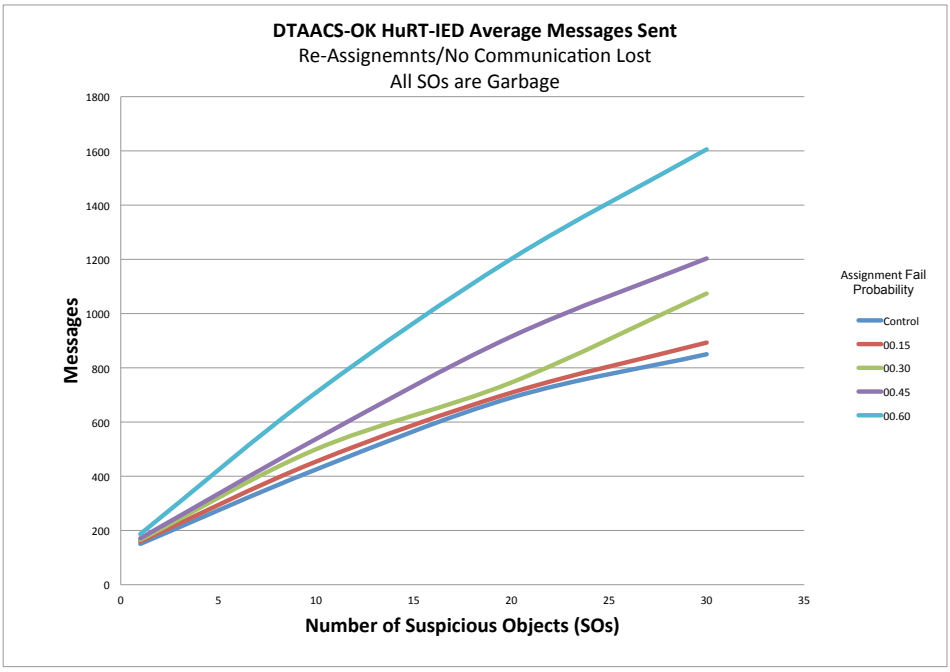

Figure 6.6: *HuRT-IED Average Message Sent with Task Reallocation all SOs are G*

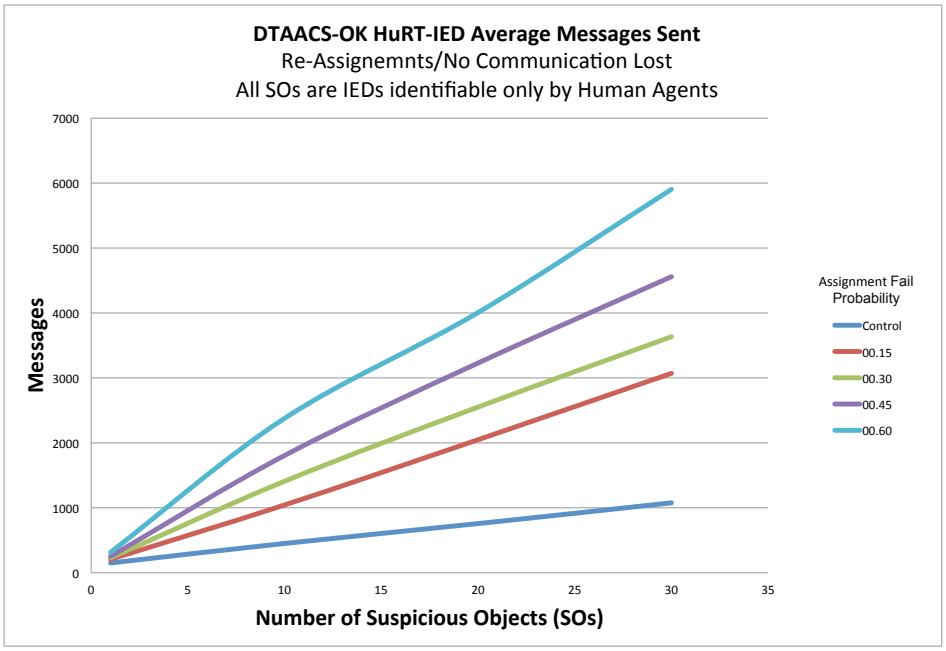

Figure 6.7: *HuRT-IED Average Message Sent with Task Reallocation all SOs are IEDs identifiable only by Human Agent*

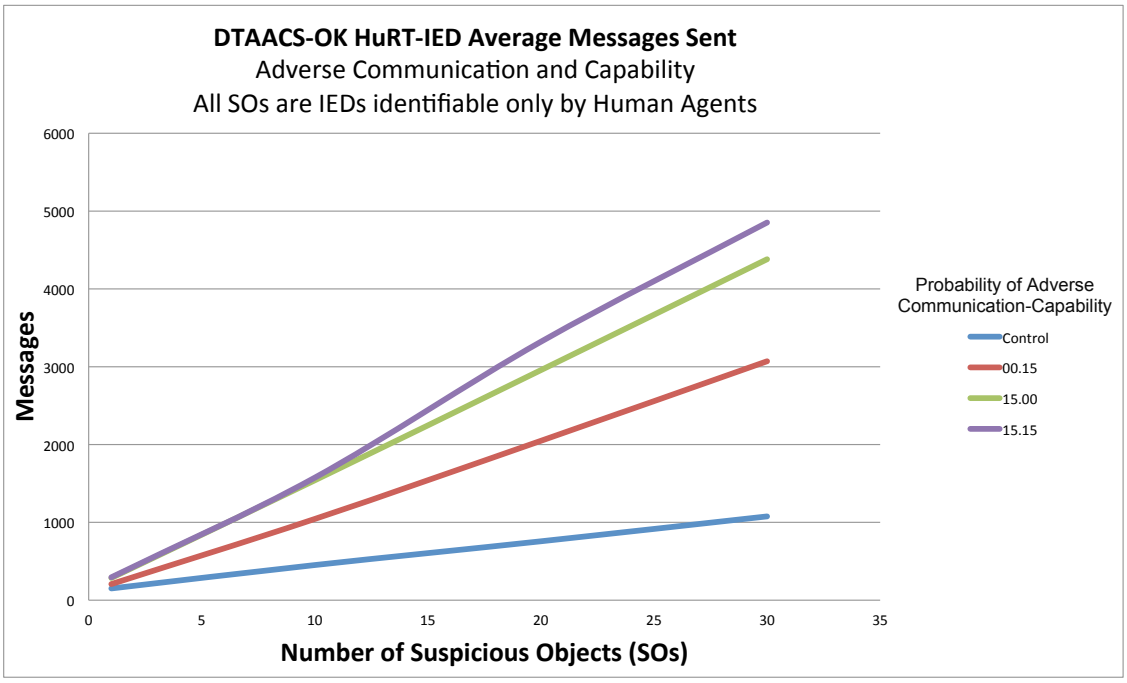

Figure 6.8: *HuRT-IED Average Message Sent with Task Reallocation 15% all SOs are IEDs identifiable only by Human Agent*

6.2 DTAACS-OK for Collaborative Assembling Objects

6.2.1 Motivation

A domain that poses interesting challenges regarding collaboration, planning, and scheduling is the domain of *flexible manufacturing control*. Jennings discusses in [30] the particular instance of a manufacturing control; the case of producing a tailored good, which the manufacturing process requires different types of objects that need to be assembled in a particular order. Jennings [30] states that the standard approach is to devise a global schedule, that in case of a delay or any failure of the entities involved in the process, an expensive re-scheduling of the process is needed. In order to show DTAACS-OK adaptability, reliability, and support for agents reorganization in case of unexpected environment changes, DTAACS-OK was integrated into a Collaborative Assembling Objects application (CAO). The CAO application was implemented as part of a research project by a KSU CIS grad student. The goal of the project was to explore a task allocation framework called DEMiRF-CF [40]. The scenario is described as a set of blocks of different types, (blocks of type A, B and C). The blocks need to be stacked one on top of the other in a predefined location in a particular order, first all blocks A, then all blocks B, and last, all blocks C. The agents in the system have the abilities to find, push, and lift the blocks. Some objects require the collaborative actions of more than one agent to be transported to the destination location. The blocks are randomly placed in the area and the destination location is randomly set.

6.2.2 Mission and Task specification

The particular mission used in this research is to stack a set of blocks that are randomly placed in a squared room. The blocks need to be stacked up in a specific location that is randomly selected, and the objects need to be stacked up in a particular order determined by the type of block. An example of a scenario of the mission is depicted in Figure 6.9. The Task Tree for the mission is depicted in Figure 6.10. The main task is to build an object

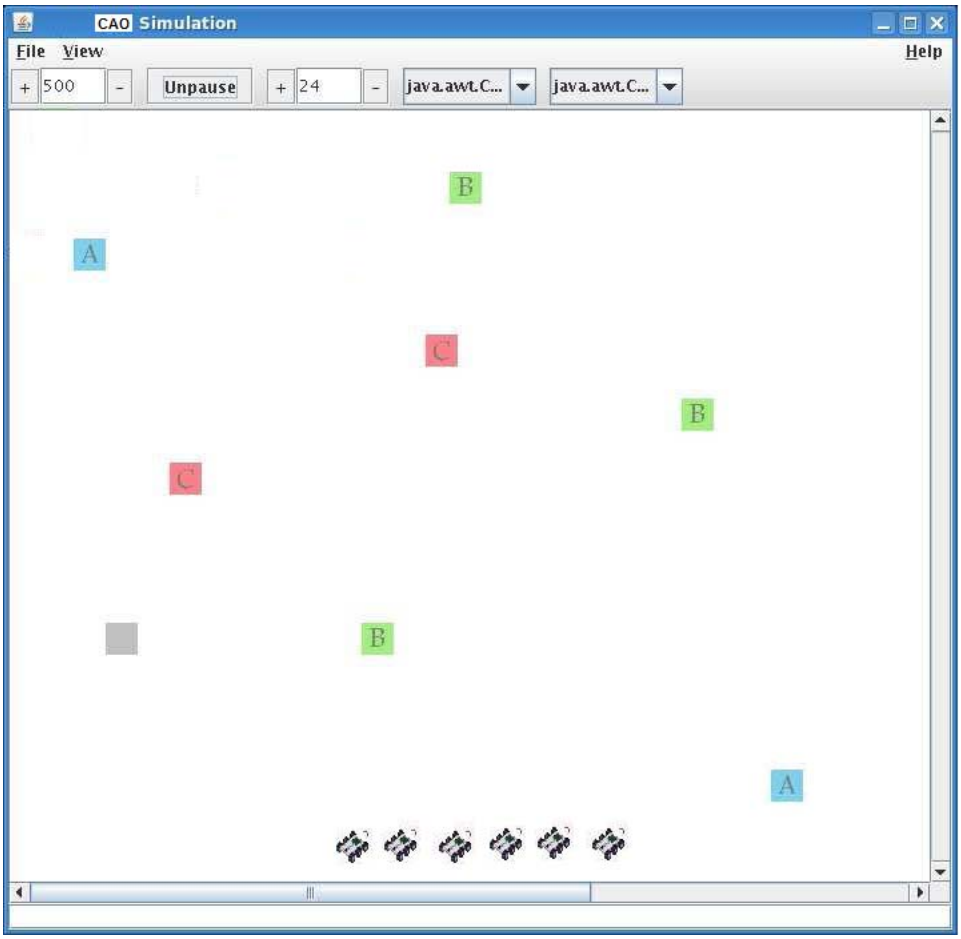

Figure 6.9: *CAO Scenario*

that is composed of different block types. We decomposed this main task into sub-tasks as follow:

Task- 0 Build Object. This is the first task that exists when the system starts.

Task- 1: Load Scenario. This task is triggered by the Build Object task. It loads the area where the blocks are placed, the location where the object is built by assembling the different blocks, the number of blocks of each type, and the expected number of robots.

Task- 2: Build-A-Blocks. This task is triggered by the Load Scenario task and it is divided into the following subtasks:

Task-2.1: Find-Blocks-A. This task takes the area to be scanned and the location where the blocks A need to be transport to. This task triggers the Transport-Blocks-A task passing the location where the block was found and the location to be transported to.

Task-2.2: Transport-Blocks-A. This task is triggered by Find-Blocks-A task when an agent finds a block of type A. The appropriate agent travels to the location where the block was found and transports the block to the destination location.

Task-2.3: Assemble-Blocks-A. This task is preceded by the task Transport-Blocks-A. Once the task Transport-Blocks-A is achieved, the appropriate agent puts the block A in the place specified by the parameter objDestA.

Task- 4: Build-B-Blocks. This task is triggered by the Load Scenario task and it is divided into the following subtasks:

Task-3.1: Find-Blocks-B. This task takes the area to be scanned and the location where the blocks B need to be transport to. This task triggers the Transport-Blocks-B task passing the location where the block was found and the location to be transported to.

Task-3.2: Transport-Blocks-B. This task is triggered by Find-Blocks-B task when an agent finds a block of type B. The appropriate agent travels to the location where the block was found and transports the block to the destination location.

Task-3.3: Assemble-Blocks-B. This task is triggered by the task Transport-Blocks-B, and it is preceded by the task Build-A-Blocks. Once the task Build-A-Blocks is achieved, the appropriate agent puts the block B in the place specified by the parameter objDestB.

Task- 4: Build-C-Blocks. This task is triggered by the Load Scenario task and it is divided into the following subtasks:

Task-3.1: Find-Blocks-C. This task takes the area to be scanned and the location where the blocks C need to be transport to. This task triggers the Transport-Blocks-C task passing the location where the block was found and the location to be transported to.

Task-3.2: Transport-Blocks-C. This task is triggered by Find-Blocks-C task when an agent finds a block of type C. The appropriate agent travels to the location where the block was found and transports the block to the destination location.

Task-3.3: Assemble-Blocks-C. This task is triggered by the task Transport-Blocks-C, and it is preceded by the task Build-B-Blocks. Once the task Build-B-Blocks is achieved, the appropriate agent puts the block C in the place specified by the parameter objDestC.

The mission is achieved when the Object is built by assembling all blocks A, all blocks B, and all blocks C in that order and in the predefined location.

6.2.3 General Scenario Description

The framework was evaluated in the following scenario. In this particular mission, the object to be build (O) was composed by blocks of type A, B, and C. The specifics of these blocks are shown in Table 6.4. Blocks of type A wight 15 lb and can not be pushed, only carried. Objects of type B weight 20 lb and can be pushed or carried. Objects of type C weight 35 lb and can be pushed or carried. Also, there were six agents initially positioned at the bottom center of the squared room as depicted in Figure 6.9. When the mission starts, and after the agents have registered with each other, the agents are instructed to start scanning the area looking for the different blocks. Each agent starts scanning the area following the greedy coverage area algorithm that selects the closest unvisited location and generates a direct path to the location, avoiding any obstacles it finds in its path. If there is more than one location with the same distance, the agent randomly selects one. The blocks were randomly distributed all over the room. For each treatment, from one to ten blocks were

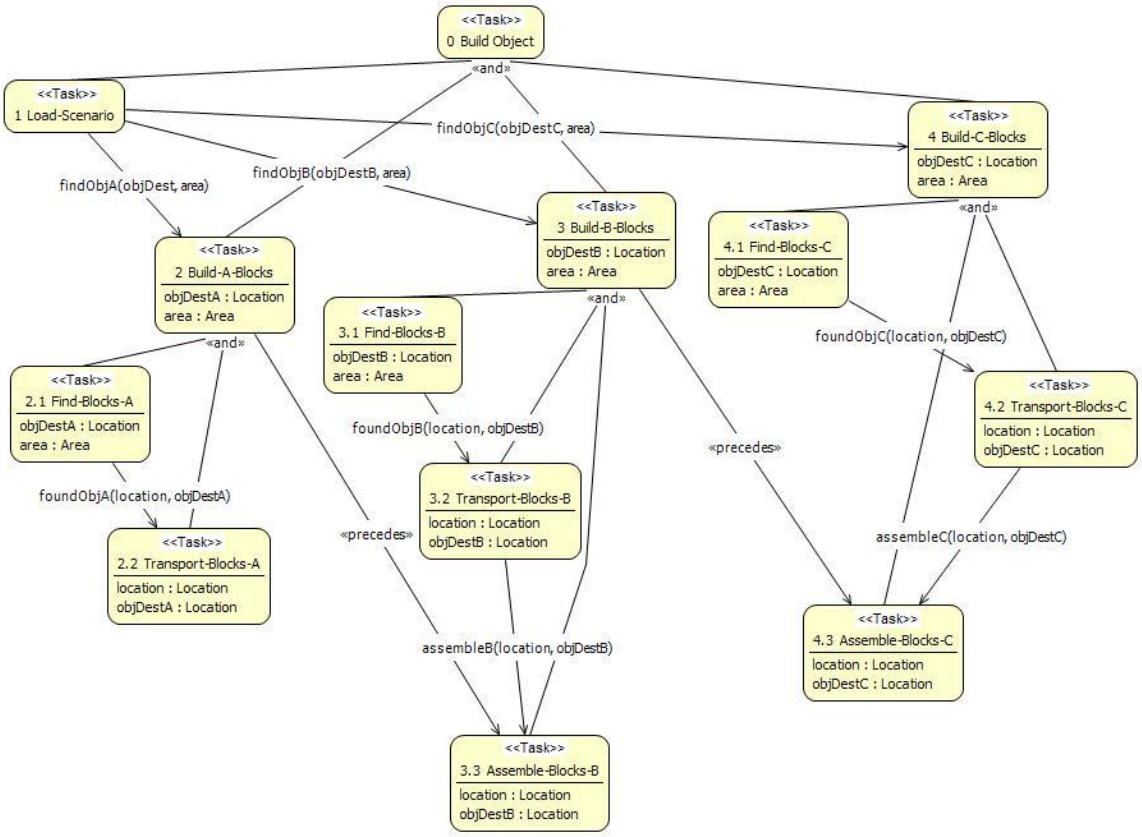

Figure 6.10: *CAO Tree Mission Representation*

placed in increments of one. The type of each block was randomly selected. To achieve the tasks in the mission, the team of agents needed to be able to scan, identify, transport, and lift the blocks. The three types of agents used in this framework evaluation are shown in Table 6.5. There were a total of six agents in each mission: Two TypeA agents (A1 and A2) were capable of finding all type of objects and posses a pushing force of 10 lb. Two TypeB agents (B1 and B2) were capable of finding all type of objects and a pushing force of 20 lb. The last two field agents were TypeC agents (C1 and C2) were capable of finding all type of objects and a pushing force of 30 lb. When agents collided, they saw each other as obstacle as well. Agents possessed a simple obstacle avoidance algorithm.

As in HuRT-IED experiment, the agents could work on one task at the time and their internal task scheduler handled the task priority defined next. Load Scenario task was exe-

Object Type	Weight	Push	Carried
A	15 Lb		√
B	20 Lb	√	√
C	35 Lb	√	√

Table 6.4: *Object Types*

Agent Type	Scan	Push	Carry and Lift
TypeA	√	10 lb	√
TypeB	√	20 lb	√
TypeC	√	30 lb	√

Table 6.5: *Robot Types*

cuted during the initialization phase with Load Scenario triggering Build-A-Blocks, Build-B-Blocks, and Build-C-Blocks. Build-A-Blocks had the highest priority, Build-B-Blocks was the second highest, and Build-C-Blocks had the lowest priority; the children task inherited the priority from their parents. When a agent detected a new task with a higher priority then the one currently executing, it saved the task progress information and swaped the task. The released task was placed in a waiting queue. The waiting task priority was incremented by one each time the agent postponed its execution because another task with higher priority arrived. In case of task failure, the released task was considered for reassignment. As in HuRT-IED experiment, for the framework evaluation, the independent variables are listed in Table 6.6 while the dependent variables are listed in Table 6.3.

6.2.4 Particular Scenario Specification

Treatment E1. E1 was the control. The independent variables message-loss and capability-degradation were set to zero. All agents had a full team broadcast communication capability. Data was collected about the number of messages transmitted, the number of task reallocations to achieve the mission, and how many turns the simulation required for the team to achieve the mission. Figures 6.11 and 6.12 show the average number of message and task

ID	Name	Description
IVar-1	Messages Loss	Since DTAACS-OK is a knowledge based solution for distributed task allocation, it is important to show how the framework behaves under less than favorable communication conditions
IVar-2	Capability Degradation	One of the main factors that determine if an agent is the most suitable for a task is the state of the set of capabilities it possesses. (Other factors may include assignment policies, or optimization goals such as load balance among agents). It is important to show how the framework performs when the agent's capabilities changes. Also, to verify that it is not necessary to send a message when a capability changes. The hypothesis here is that when a capability changes, the change may not be severe enough to change the suitability of the agent form most suitable to a lower level, or vice versa. (The last condition may occur when the distance from the current agent's position to a destination determines who gets the assignment).
IVar-3	Number of Blocks A, B, and C	To demonstrate how the framework performs in relation to scalability, different numbers of blocks to search, transport, and assemble were placed.

Table 6.6: *CAO Independent Variables*

allocation results over a hundred runs for scenarios with three to ten blocks randomly set to type A, B, or C.

Treatment E2. For E2 the assumption of perfect communication conditions was relaxed. DTAACS-OK was tested under 0%, 15% and 30%, and 60% probability of message loss with no communication delay and full broadcast capability. There was no capability degradation for this treatment. Figure 6.11 shows the results for average number of messages over a hundred runs for the same condition regarding blocks A, B, and C as in E1.

Treatment E3. In the third treatment, the framework was stressed by introducing agent capability degradation to the point that agents were not able to execute and achieve their tasks. The number of messages sent for task failure were thus counted. The agents had full

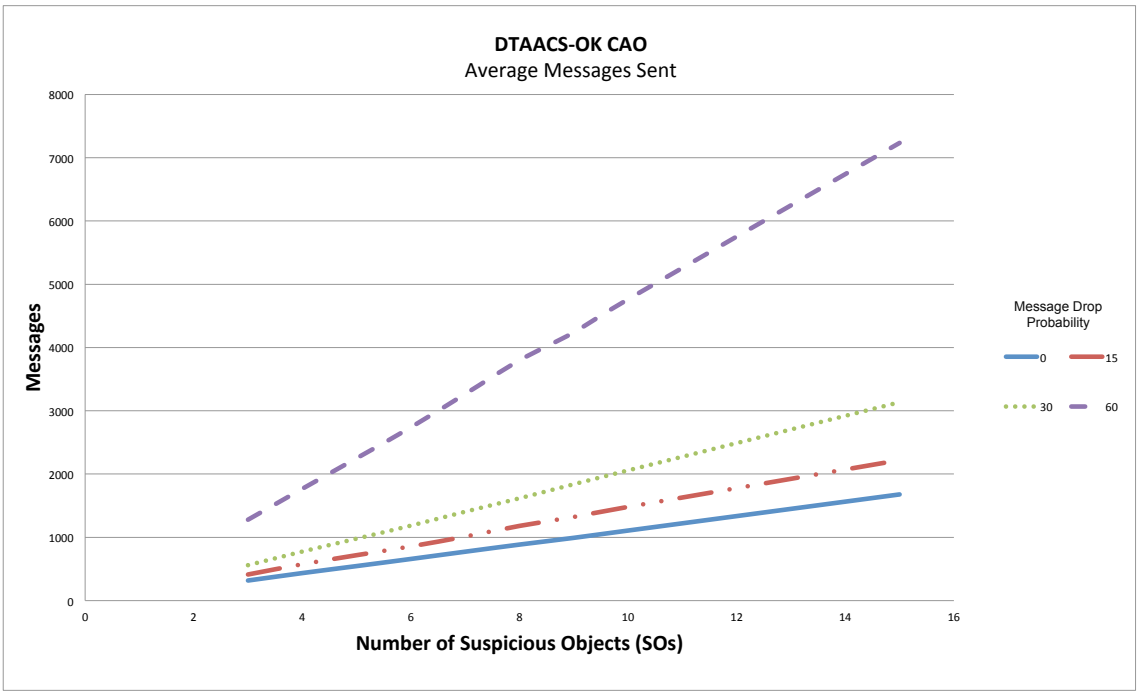

Figure 6.11: *CAO Average Messages Sent for 0%, 15%, 30%, and 60% Probability of Message Drop*

broadcast communication, no delays and no message drops, and there was at least one agent in the organization capable of executing all tasks that remained. For this treatment, when the agent's sensor and gripper capabilities de-gradated, they did not recover. Results for the average number of task allocation are shown in Figure 6.12, and for the average number of messages sent due to agents' failing a task over a hundred runs are shown in Figure 6.13.

Treatment E4. In treatment E4 a more realistic simulation was used to test DTAACS-OK. E4 is combination of E2 and E3 where communication is not perfect and agents' capabilities may fail in some degree and cause reallocation of tasks. Figure 6.14 shows the results of a hundred runs for communication conditions where there was 15% probability to drop a message and 15% probability that an agent fail to achieve a task, and therefore a reallocation occurred. Figure 6.15 shows result for 30% in both communication and capability degradation probability.

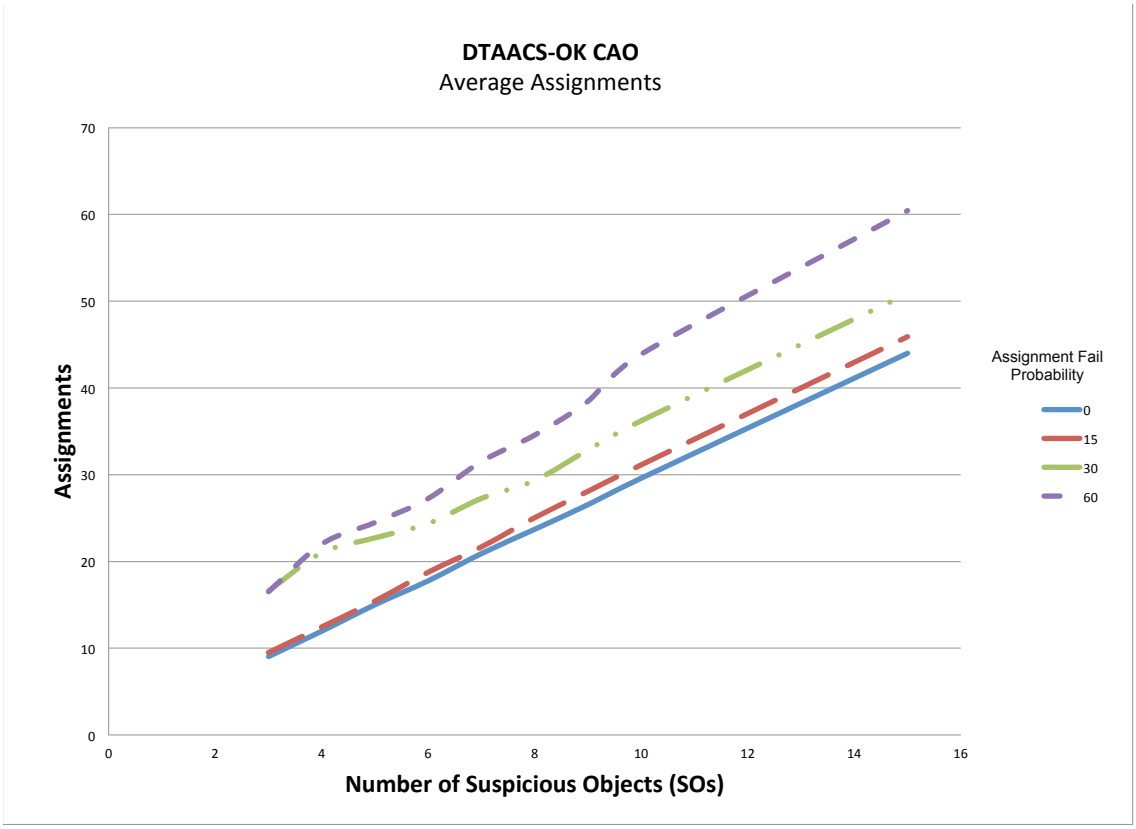

Figure 6.12: *CAO Average Assignments for 0%, 15%, 30%, and 60% Probability of Task Failure*

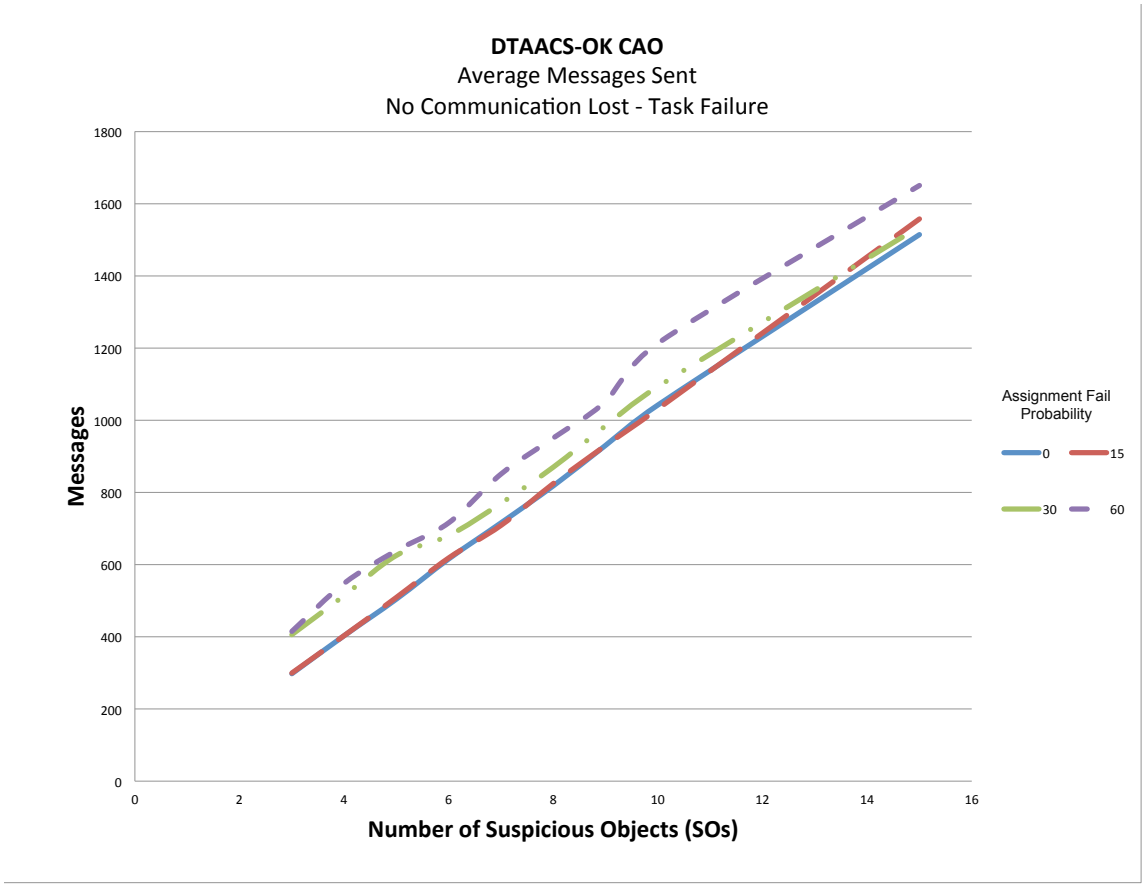

Figure 6.13: *CAO Average Messages Sent Under Task Failure*

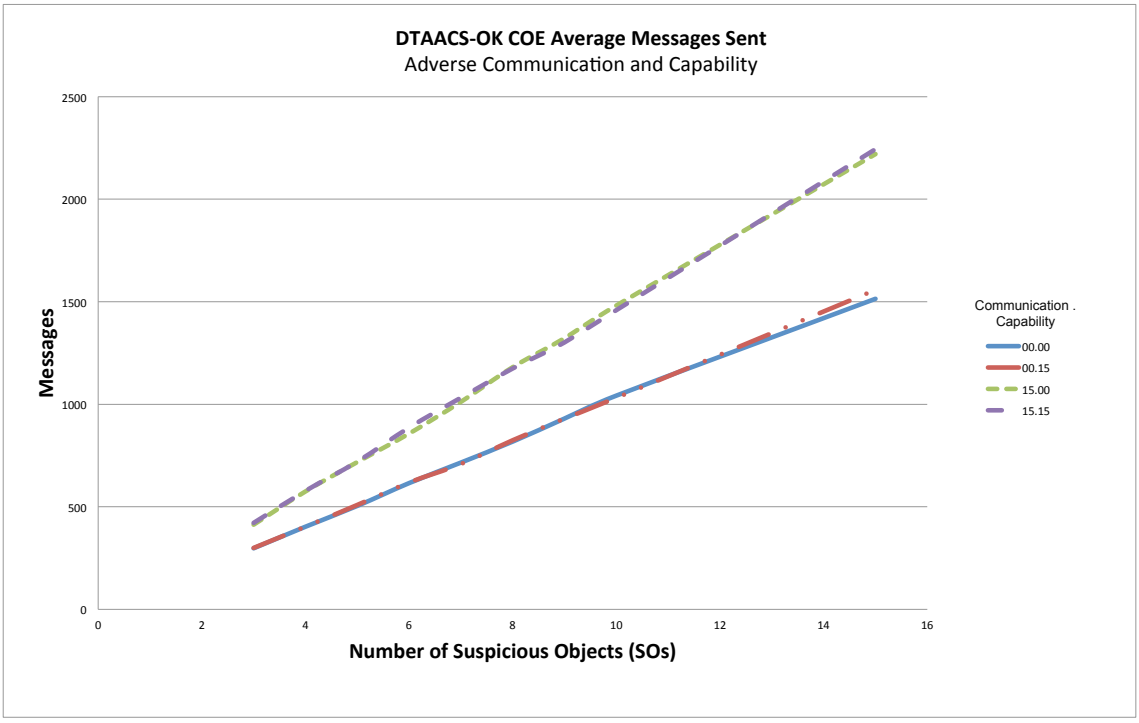

Figure 6.14: *CAO Average Assignments Under Task and Communication Failure (15%)*

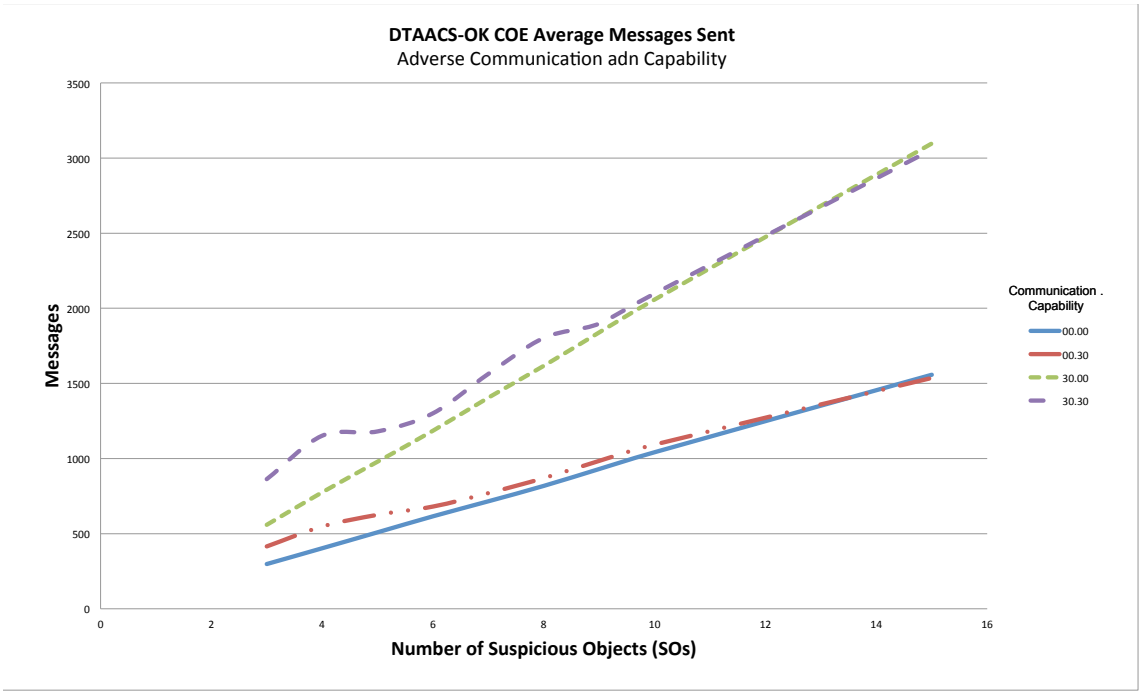

Figure 6.15: *CAO Average Messages Sent Under Task and Communication Failure (30%)*

6.3 DTAACS-OK versus DEMiRF-CF

In this section, a comparison between DEMiRF-CF[41] and DETAACS-OK is presented. These two solutions address the task allocation problem in MAS. The comparison is offered as a relative assessment of the performance of the frameworks in relation to communication cost in achieving the mission in the Cooperative Assembling Object (CAO) application. The result compared are the ones obtained in the empirical evaluation of DTAACS-OK as described in Section 6.2. The result for DEMiRF-CF for the COA application were obtained by implementing the experiment described in Section 6.3.2.

6.3.1 Motivation

DEMiRF-CF[41] is a framework that implements the Incremental Multi-Robot Task Selection (IMRTS) approach [42]. As discussed in Section 7.1.3, IMRTS is a marked based solution for the task allocation problem which has similarities with the solution offered by DTAACS-OK. Table 6.7 shows the similarities between these two approaches.

No.	Similarity
1	Both address the task allocation problem in cooperative MAS
2	Both address complex tasks
3	Both address tasks relations like AND, OR, and precedece
4	Both form coalitions for complex task if needed

Table 6.7: *DTAACS-OK and DEMiRF-CF Similarities*

6.3.2 General Scenario Description

The scenario for which DEMiRF-CF for the COA application was evaluated was the same as the one described in Section 6.2.3.

6.3.3 Particular Scenario Specification

Treatment E1. E1 was the control. The independent variables message-loss and capability-degradation were set to zero. All agents had a full team broadcast communication capability. Data was collected about the number of messages transmitted to achieve the mission. Figures 6.16 shows the average number of messages sent over a hundred runs for scenarios with three to fifteen blocks randomly set to type A, B, or C.

Treatment E2. For E2, the assumption of perfect communication conditions was relaxed. DEMiRF-CF and DTAACS-OK were tested under 15% probability of message loss with no communication delay and full broadcast capability. There was no capability degradation for this treatment. Figure 6.17 shows the results for average number of messages sent over a hundred runs for scenarios with three to fifteen blocks randomly set to type A, B, or C.

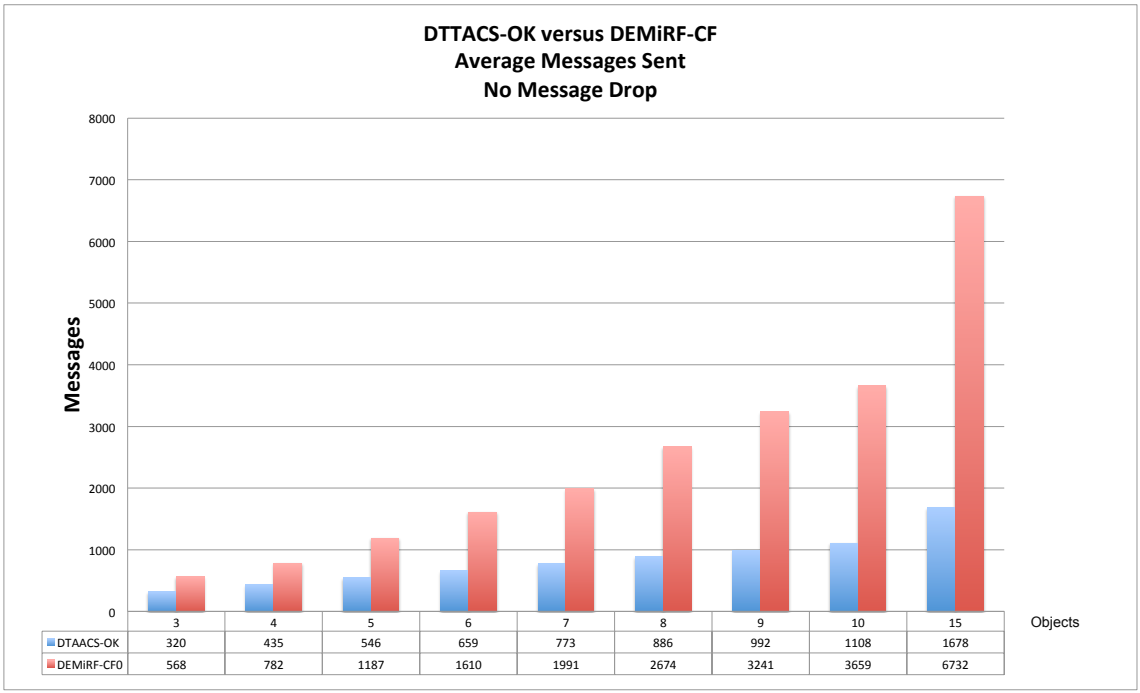

Figure 6.16: *DTAACS-OK versus DEMiRF-CF CAO No Message Drop*

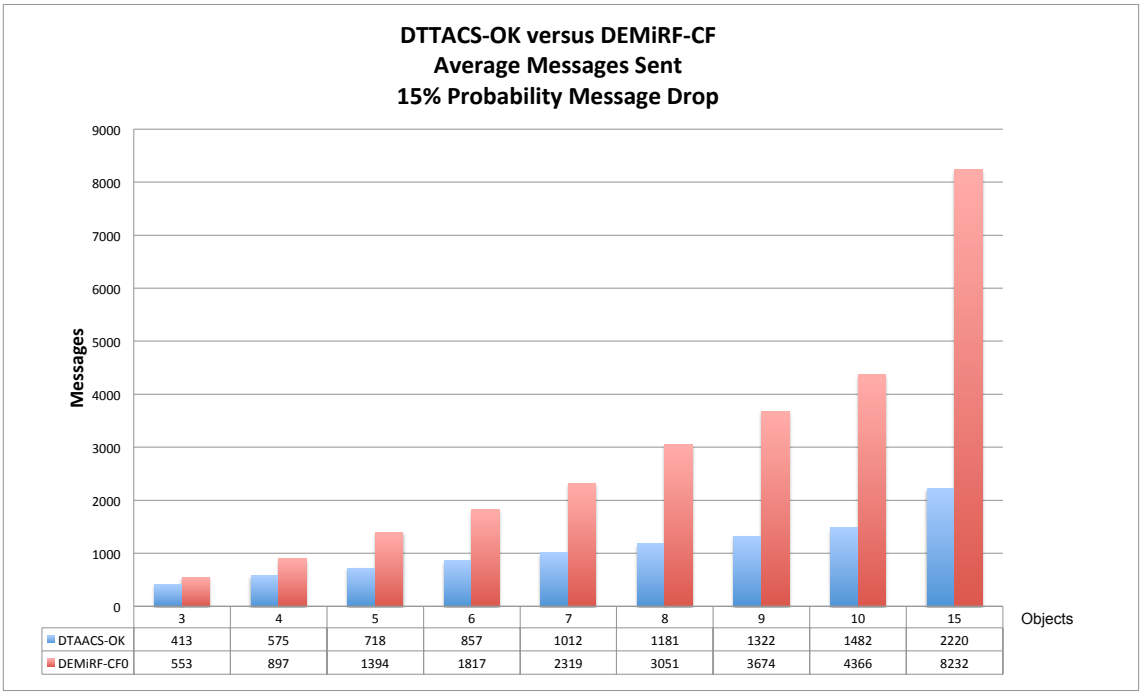

Figure 6.17: *DTAACS-OK versus DEMiRF-CF 15% Probability Message Drop*

6.4 Summary

In this chapter, the empirical evaluations result for DTAACS-OK were presented. DTAACS-OK was evaluated using two applications: (1) HuRT-IED is an application where a team of robots have a common mission to clear an area of IEDs; the team is monitored and supported by a human agent, and (2) CAO is an application where a team of robots have a the mission to assemble an object compose of different types of smaller objects in a collaborative way. The goal of these experiments were to evaluate DTAACS-OK mainly regarding communication cost and how the framework reacted under agents failing to achieve a task due to capability degradation.

DTAACS-OK was also compared to DEMiRF-CF, a framework that implements IMRTS. DEMiRF-CF was implemented by a graduate student in CIS at Kansas State University. The frameworks were compared regarding the number of messages sent under conditions with no communication loss and 15% chances of dropping a message.

Chapter 7

Related Work

The Task allocation problem has been studied extensively in areas such as Operational Research [53], Economics [22] and Computer Sciences [4, 9, 37, 44]. Solutions proposed in computer sciences, particularly in MAS, can be categorized into two groups: *centralized* and *distributed* approaches. In [50], van der Horst and Noble addressed *when* and *where* to use a centralized or distributed approach. They concluded that answers to these questions can be derived from system characteristics, such as parameters to optimize and the number of agents in the system. However, both approaches have application domains for which are more suitable to be used. Zhang et al. [56] presented five categories of existing task allocation approaches:

- *Fully centralized approaches.* In these solutions, the central allocator possesses a model of all members of the system and calculates the cost for each when allocating tasks. These approaches require full communication with all team members and full knowledge of the member state and task status; the allocator is a single point of failure.

- *Centralized auctions.* In these solutions, an auction coordinator determines the participant with the lowest cost after receiving the cost of executing a task calculated by each participant. Because the cost is calculated by each participant, the coordinator does not have to keep a model of each participants. A bidding protocol must be executed, which presents communication and computational disadvantages compared to

the *fully centralized* approaches.

- *Distributed auctions.* In general, a distributed auction approach is the result of executing *local* centralized auctions simultaneously. That is, several auctions can be executed in parallel so that a participant can bid in more than one auction. These solutions, as in centralized auctions, do not require the coordinator to keep a model of the participants, but inherently yield suboptimal task allocations when a global cost function is considered.

- *Completely distributed approaches.* These approaches do not require direct communication among system members; each determines actions based on observations of the environment where they are situated on. These solutions are robust against communication failures but yield suboptimal task allocation results.

- *Hybrid approaches of distributed actions and emergent coordination.* This approach aims to benefit from the advantages of implicit negotiation and explicit negotiation, but can yield highly suboptimal task allocation solutions.

In this chapter, relevant work related to the *task allocation problem* in MAS is discussed. One broadly used approach is the market-based approach discussed in Section 7.1. Another approach is one that tackles the problem as a Markov Decision Problem (MDP); MDP solutions proposed in the literature are discussed in Section 7.2. In Section 7.3, other approaches that do not fall into the two previous sections are discussed. Another approach addressing the task allocation problem is based on the coalition formation as proposed by Sheory and Kraus in [46] and discussed in Section 7.4.1. Vig and Adams in [51] and [52] present an improvement to the Sheory and Kraus' solution [46], presented in Section 7.4.2. These two solutions inspired coalition formation algorithms proposed in DTAACS-OK, additional coalition-based approaches are discussed in Sections 7.4.3 and 7.4.4. Work related to the allocation of tasks to a team of agents is presented in Section 7.4.

7.1 Market-Based Approaches

In Market-Based Approaches, the allocation of tasks to agents is the result of a contract net protocol [1]. A requirement for such approaches is that each agent must have the means to calculate the cost of executing the anticipated task. The assumption is also that agents have knowledge of all tasks to be allocated and, when an agent identifies a desired task, the agent begins an auction protocol with the rest of the agents in order to discover other agents interested in the task and the correlating cost of execution. After receiving all bids, the auction manager determines and notifies the agent with minimal cost by sending an award message. Some solutions proposed in the literature and their particular characteristics are discussed in the following section.

7.1.1 M+

M+ [16] is a decentralized protocol for (a) planning, (b) online task-decomposition, and (c) allocation of loosely coupled tasks in multi-robot environments. M+ is designed to be implemented as a task layer on top of an action layer. The main functionality of M+ is a negotiation mechanism, which receives a list of tasks from a higher level called, Mission Layer. The M+ protocol is implemented by two entities, *task planner* and *task supervisor*, and it utilizes GraphPlan as the standard planner. The M+ protocol performs three primary activities:

1. *Task allocation.* This module is in charge of task refinement and task allocation. It also embeds the negotiation mechanism that allows a robot to choose incrementally the best task to be executed.

2. *Cooperative reaction.* This module is invoked when a failure occurs during task execution.

3. *Execution.* This module is in charge of task execution control as well as distributed synchronization between robots' tasks and actions.

M+ is suitable for systems in which tasks are loosely coupled, but it does not manage joint tasks and does not generate cooperative plans during task execution. Online task decomposition is based on predefined goals that compose a task and goals that consist of a set of actions. However, allocation occurs at task level and does not take advantage of allocating actions when the robot is not able to perform a particular action. A task example includes transfering a container c from station s_x to station c_y. Action examples include *pick-up-from-station, put-down-on-station, go-to.*

7.1.2 TraderBots

TraderBots [19] is an architecture for coordinating a multi-robots team to achieve a common goal in a dynamic environment. TraderBots is a market economy approach that allows cooperation and competition in an opportunistically way. When a task is announced in an auction, the participant self-interested robots compute bids based on expected profit on the task. The robot with the lower/higher bid is selected and a contract is established. The winner robot can take the role of auctioneer and reallocate the task or a portion of the task. Traderbots is a market-based approach that requires a revenue *(trev)* function and a cost *(tcost)* function; the *(trev)* function maps task outcomes into revenue, and the *(tcost)* function maps possible schemas for executing the task into cost values. These functions may be complex, and they are application specific. In Traderbots, *cooperation* occurs when two robots complement each other (one robot provides a service to the other) and when they execute a task together, thus producing a higher profit *(tcost - trev)* than would be attained if executing the task individually. *Competition* occurs when the profit of one robot is negatively affected by the presence of another robot. TraderBots is a general market-based approach that does not consider whether the team is homogeneous or heterogeneous; however, TraderBots is designed to handle simple tasks, single robot tasks, and tasks in which robots do not work in close coordination. Adversarial domains, cooperative recovery from

failure, and the use of partially damaged robots is not addressed [19]. An extended version of Traderbots is presented in Section 7.1.3, address limitations and future work listed in [19].

7.1.3 Incremental Multi-Robot Task Selection

Incremental Multi-Robot Task Selection (IMRTS) [41, 42] is an extension of Traderbots which introduces complex tasks to Traderbots. A *complex task* is defined as a hierarchical tree of tasks in which the root of a tree is a higher abstraction of the subtasks [42] . These tasks and subtasks are related by AND and OR relations, depending on the application and degree to which the tasks are coupled. A precedence relation may be exist between tasks. In IMRTS, the task allocation problem is approached as a scheduling problem and the problem is formulated as a *resource constrained project scheduling problem* from Operational Research (OR). The IMRTS approach and the framework of this dissertation (DTAACS-OK) have similarities, such as addressing complex inter-related tasks and tight-coupled tasks. However, IMRTS is a market-based approach, while DTAACS-OK is a distributed organization *knowledge based* approach. Another similarity is that coalitions formed in IMRTS follow the algorithms proposed in [45], while DTAACS-OK coalition formation algorithms are also inspired by [45] but are modified to consider replicated organization knowledge in the decision-making process. Also, in the Incremental Multi-Robot Task Selection approach, a robot can compute its own task decomposition, but the decomposition steps are not specified.

In the following section, other approaches in the literature that model the task allocation problem, such as a Markov Decision Problem, are presented.

7.2 Markov Decision Problem

For systems in which the environment is uncertain and dynamic, Markov Decision Problem (MDP) models offer a way to overcome those characteristics [47]. When an MDP model is

used for task allocation in a MAS, an agent chooses an action available in that state, receives a reward, and probabilistically changes to a different state. In Section 7.2.1, an MDP-based approach for the task allocation problem addressed as a scheduling problem is presented. Another MDP approach with the ability to handle spontaneous tasks is discussed in Section 7.2.2.

7.2.1 Decentralized Dynamic Task Allocation

The approach proposed in [12] targets MAS where system tasks have a hard deadline and specific execution time. The solution proposed by Chapman et al. [12] handles task allocation with a scheduling mechanism modeled as a Markov game with complete information. The known facts are: (1) current system states, (2) uncertainty of future states, (3) finite set of strategies possessed by the agents, and (4) specific number of time steps for which the agents play. Markov stochastic games are intractable, and the authors use a sequence of potential games to tackle intractability. Potential games are a subclass of noncooperative games that possess two practical properties: (1) every finite potential game possesses at least one strategy equilibrium, and (2) potential games possess the *finite improvement property*, meaning that any sequence of unilateral improving moves converge to a Nash equilibrium in finite time. The task allocation model comprises a set of states that define a set of tasks. Each task has a deadline, a processing unit, and a task function. The task allocation model also includes a set of agents, each agent possesses a strategy space, each strategy composed of a sequence of tasks to attend and a utility function, a state transition function, and a global utility function. The allocation of tasks in this approach is the result of the execution of a *distributed stochastic algorithm*, which is a greedy local search algorithm. This approach addresses two challenging task constraints: *hard-time deadlines* and *specific execution-time*. However, not all application domains present these constraints, and the communication cost and solution optimality are not discussed in this paper. Another MDP approach with the ability to handle spontaneous tasks is discussed in Section 7.2.2.

7.2.2 Modeling Task Allocation Using a Decision Theoretical Model

Abdallah and Lesser tackle the task allocation problem in [3] as a decision problem that a *mediator* faces when a task arrives for which the agent cannot independently execute. In the task allocation decision problem, the set of tasks available to a mediator at time t is the set of actions A_t. Abdallah and Lesser define a mediator as an agent that discovers a task that it cannot execute. The mediator decomposes the task (if necessary), finds agents with capabilities required by the task, and negotiates with the agents found the required commitment in order to execute the task or subtask. This mediation process is modeled as a decision problem. A main characteristic of a decision problem is that it can handle tasks that appear randomly, called the *randomly available actions* property. The authors model this decision problem as a Semi Markov Decision Problem (SMDP) in order to exploit the randomly available actions property that an MDP fails to exploit. In [3], emphasis is place on the higher-level decisions of *whether* to start the allocation of a task, and *when* to stop the allocation process even if various subtasks have been allocated. Abdallah and Lesser assume that a negotiation protocol is used, such as the Contract Net Protocol [1], and ignore the implications of communication among agents and the computational cost of executing protocols. The authors present an SMDP model with the capability of handling serialized contracting of subtasks and then extends the model by incorporating a *Concurrent Action Model* [**reference**] in order to allow the mediator to handle the concurrent mediation of tasks.

Abdallah and Lesser's solution [3] provides to the agents decision mechanism for when to stop the execution of a task t_i and swap it for a new task t_j that arrives and how to reallocate the task t_i and subtasks (if it is decomposed) to other agents. However, the decision of which agent is most suitable to the tasks still relies on the Contract Net Protocol.

7.3 Other Approaches

The approaches discussed in previous sections and DTAACS-OK can be categorized as *defined a priori by a designer* [21]. However, an organization structure can also be set through an *emergent organizations* approach in which the organization structure is set *a posteriori* [21]. Emergent organizations are formed by reactive agents which, through their interactions and behavior, define the relationships among them and form an organization, as is the case of *Alliance*[35], discussed in Section 7.3.1. In all task distribution approaches, *knowledge* concerning agents and tasks, regardless of how the knowledge is acquired and maintained, is a key factor in making decisions as to *who* is assigned to *what*. In Section 7.3.2, an approach that incorporates a *decision making* model based on organization knowledge is presented.

7.3.1 Alliance

Alliance [35] is an architecture for fault tolerant MAS in which robots cooperate to achieve a common mission. The mission is composed by loosely coupled tasks that may have ordering dependencies. Alliance is a distributed behavioral-based architecture. Alliance aims to provide robots in the system with a behavior mechanism that allows them to be robust, reliable, flexible, and coherent under dynamic and unpredictable environments. Actions taken by the robots are determined by a mechanism (*motivational behaviors*) based on a mathematical model that considers the following information:

- Mission requirements
- Other robot activities
- Current environment conditions
- Robot's own internal states

Motivational behaviors mechanism allows the robots to react and adapt to different events and to work on a task as long as progress is made toward mission achievement.

Motivational behaviors receive input from sensory feedback, inter-robot communication, inhibitory feedback from other active behaviors, and internal motivations. Internal motivations modeled in Alliance are *robot impatience* and *robot acquiescence*, discussed below.

Robot Impatience. This motivation triggers the robot to handle situations in which other robots, besides itself, fail to perform a task. For each task, robot impatience increases rapidly if that task is not being accomplished by other robots; *robot impatience* increases at a slower rate if the robot is aware of another robot working on a task which is expected to be accomplished in a predefined period of time. Alliance requires every robot to broadcast its progress on pursuing a task.

Robot Acquiescence. This motivation indicates that the robot is failing to achieve a task. Each robot maintains an indicator (w) for the current task which increases over time; w reflects the *willingness* of the robot to stop working on the task. Two cases may occur: (1) the robot realizes it is not capable of achieving the task (indicator crosses threshold) and (2) a robot B informs the robot A that it has taken over the task so robot A decides to release the task based on its w value.

This approach explicitly addresses the fault tolerance of situated MAS but, as mentioned by Parker in [35], sacrifices efficiency for robustness and autonomy. Alliance requires that agents broadcast the status of the task they are executing, otherwise task reallocation would have a high rate. Alliance does not address coalition formation.

7.3.2 Distributed Task Allocation in MAS based on Decision Support Module

The Task Allocation Decision Maker (TADM) [20] is a module part of a framework called *Distributed MultiAgent Intelligent System.* The main purpose of the TADM module is to choose which agent is the most suitable for a selected task. in this approach, each agent executes the algorithm independently and randomly selects a task to allocate. The module contains four main components.

1. Decision Knowledge Management
2. Data Organizer
3. Decision Maker, which is subdivided into two modules:
    - Task Allocation Decision Maker
    - Coordination Mechanism Selection Decision Maker
4. Evaluator Module. This module (a) executes actions selected by the Decision Maker module and (b) evaluates the results of the executed actions.

In the *decision making* process, Granular Computing and Rough Set Theory (RST) are utilized. Granular Computing combines the advantages of using RST tools with the benefits of granulation. Rough sets are used to address incompleteness, imprecision, and subjectivity of the information that agents possess and are present in MAS. TADM incorporates a Granular Rough Model with four main concepts:

1. Decision Table
2. Set Approximation
3. Data Reduction Phase
4. Rule Generation Phase

The TADM approach is similar to DTAACS-OK because both approaches use knowledge about the system, but since the information is updated independently by each agent, TADM is not *distribute-consistent* among the agents when allocation decisions are made. Information update actions still must occur and, although flexibility is gained by doing them independently, TADM may lead to work duplication and impact the efficiency and optimality of the solution. In this approach, the agents randomly select the next task, but TADM does not specify how tasks priorities are handled.

7.4 Coalition Formation and Task Allocation

7.4.1 Task Allocation via Coalition Formation

Shehory and Kraus present a task allocation solution via coalition formation [46]. The authors acknowledge that the problem is NP-Complete and propose a heuristic approach based on an algorithm presented by Chavat [13]. Shehory and Kraus tackle the problem by finding the smallest group of agents for the tasks. The greedy algorithm consists of two stages. In the first stage, the agents calculate all possible coalitions up to a value k in which they can participate and calculate a coalition value. In the second stage, the agents communicate in order to update the coalition value, agree as to what coalition they will participate, and form the coalition. In the Shehory and Kraus paper, disjoint and overlapping coalitions are addressed. Although the authors point out the generation of small coalitions (*small* k) is more advantageous, Shehory and Kraus do not specify how this key value is calculated. In the coalition formation algorithms presented in DTAACS-OK, the key value k is incrementally calculated until a valid coalition is found (see Chapter 5). The DTAACS-OK coalition formation algorithms incorporate changes to Shehory and Kraus' algorithms [45] proposed by Vig and Adams [52] to address the fact that, in multi-robot systems, resources cannot be transfer from one robot to another. The Vig and Adams solution is discussed in Section 7.4.2.

7.4.2 Multi-Robot Coalition Formation

Vig and Adams present [51, 52] two modifications to the algorithms proposed by Shehory and Kraus in [46]. The modifications goal is to make the algorithms proposed in [46] more suitable to Multi-Robot Systems, in which resources cannot be easily transferred from one robot to another. The first modification proposed by Vig and Adams in [51] is a *Task Allocation Matrix* (TAM), which is a representation of the tasks as a capability matrix such that each cell represents a resource pair required by each task. A cell is set to 1 if the pair of resources must reside on the same robot. A cell is set to 0 if the resources must reside

on separate robots. If the location of the pair of resources is not relevant, the cell is set to X. The matrix TAM determines weather or not a candidate coalition is valid. The second proposed modification attempts to avoid the generation of unbalanced coalitions, defined as coalitions in which a particular robot contributes a majority of the resources, making the coalitions less fault tolerant.

In DTAACS-OK, the coalition formation algorithms consider resources locations as proposed in [46]. In the current status of DTAACS-OK, coalition balance is not considered.

7.4.3 Bayesian Model-Based Coalition Formation Approach

A Bayesian model-based reinforcement learning framework for repeated coalition formation under uncertainty is presented by Chalkiadakis and Boutilier [11]. The authors claim that most models proposed for coalition formation assume agents have knowledge of the capabilities of other agents, which is not always true for real scenarios. Chalkiadakis and Boutilier propose the utilization of repeated interaction between agents in order to acquire information to improve decision making when forming coalitions. The Bayesian reinforcement learning model (RL) allows the agents to improve their decision making process to form teams with known agents (exploitation) or new agents (exploration). The Bayesian RL model is formulated as a partially observable Markov decision process (POMDP). The agents, based on the solution to this POMDP, generate policies that evaluate actions for immediate gains and potential coalitions. Because POMDPs are computational intractable, the authors propose approximations to allow the effective sequential generation of policies.

7.4.4 Building Coalitions Through Automated Task Solution Synthesis

Parker and Tang [36] present a distributed version of the ASyMTRe algorithm, ASyMTRe-D, in order to form coalitions. The primary differences in ASyMTRe compared to other approaches are the change from typical abstraction of a *task* to a biologically inspired *schema* and a mechanism in order to address multi-robot tasks. The basic block in ASyMTRe is

the computational model called a *schema*, which defines a list of input and output ports in addition to a behavior, thus defining how the input is processed in order to generate the output. Robot collaboration occurs when the output of one schema in one robot is connected autonomously and dynamically to the input ports of another schema in another robot. Connections are established based on knowledge requirements of each task. Coalitions in ASyMTRe are formed by solving the configuration problem defined by a set of robots R, a set of tasks T, and a utility function U. The set T is composed of a set of schemas that define the group-level task to be achieved. The connections are regulated by a set of connection constraints. ASyMTRe defines a *potential solution* as one way to connect schemas in a particular robot in order to partially fulfill a task, to completely fulfill a task, several potential solutions may be required. With multiple potential solutions, the *utility function* helps determine the quality of each potential solution. When searching for a set of potential solutions to completely fulfill a task, a *potential configuration space* (PCS) is generated which can grow exponentially, to reduce the size of this PCS, a policy is introduced specifying that only one entry for each equivalent class of schema is considered. The authors also provide analysis of the soundness, completeness, and optimality of the algorithms.

Chapter 8

Discussion And Conclusion

The work presented in this dissertation solves the task allocation problem in OMAS. The solution proposed, DTAACS-OK, is a distributed approach that takes advantage of the system status information shared among agents. This system information is shared in most of the solutions found in the literature, but it is not used as in DTAACS-OK. DTAACS-OK aims to provide a low communication cost solution to the task allocation problem in OMAS. The allocation of tasks is based on identical organizational knowledge being possessed by all agents in the organization when of processing the same event. This solution is in contrast to current solutions in that it does not use marked-based task auctions, or is not based on motivational behaviour as in the Alliance [35] approach. A more elaborate description of the proposed solution is presented in Section 8.2

The Task Allocation Problem is not new and has been addressed in different areas of science and economics; yet, it is a problem that still attracts researchers as the rapid development of technology challenges them to exploit the resources and to deploy computational systems where it was difficult in the past. A brief summary of some approaches found in the literature is described below.

8.1 Prevailing and Relevant Solutions

The solutions proposed for the task allocation problem in computer sciences, and particularly in MAS, fall into two main categories: *centralized* and *distributed* approaches. Most of

the latest and popular approaches fall into *Centralized auctions* and *Distributed auctions* as defined by Zhang et al. [56] (See Chapter 7). Other solutions take a different approach and model the task allocation problem as Markov Decision Problem. Yet, other approaches are behavioural-based and emergent organizations are formed by reactive agents. The solutions relevant to this research are those that uae distributed auctions and are briefly reviewed below.

M+. A popular marked-based approach in the late 1990s was **M+** [16], which is designed to be implemented as a task layer on top of an action layer. M+ performs three main activities: (1) Task allocation, which includes a negotiation mechanism for incrementally select the best task to be executed, (2) Cooperative reaction, which is invoked when a failure occurs and (3) Execution that is in charge of the actual execution of the task. The two main disadvantages of M+ are that: (a) M+ does not handle joint tasks and (b) M+ does not generate cooperative plans during the execution of the tasks.

TraderBots [19] was proposed in 2004 to improve certain areas of M+. TraderBots continues using a marked-based approach, but also offers a mechanism to allow cooperation and competition in an opportunistic way. However, it is designed to handle simple tasks, single robot tasks, and tasks in which robots do not work in tight coordination.

IMRTS. In 2007 Sariel et al. [42] proposed IMRTS, an incremental multi-robot task selection approach as an extension of TraderBots. IMRTS introduces complex tasks and incorporates algorithms to form coalitions needed for complex tasks. The approach proposed by Sariel et al. [42] and the framework produced in this dissertation (DTAACS-OK) have similarities, such as addressing complex inter-related tasks and tight-coupled tasks. However, the approach taken by Sariel et al. is market-based while DTAACS-OK takes a distributed knowledge-based approach.

Decentralized Dynamic Task Allocation. This solution was proposed by Chapman et al. [12] and addresses the task allocation problem with a scheduling mechanism modeled as a Markov game. Unfortunately, the Decentralized Dynamic Task Allocation approach is

limited to certain domains where the tasks' execution time, and tasks' deadlines are known in advance.

Alliance. Alliance provides the robots in the system with a behavior mechanism to achieve a common mission in a dynamic and unpredictable environments. The motivational behaviour mechanism receives input from sensors, communication with other robots, and internal motivation model. The internal motivation includes *robot impatience* which is a measure of other robots failing to execute a task. The other internal motivation is *robot acquiescence* which indicates that the robot is failing to achieve a task. Alliance is a solution to the task allocation problem that sacrifices efficiency for robustness and autonomy, and does not address coalition formation.

8.2 DTAACS-OK: The Framework

This section reviews the framework proposed in this dissertation to address the task allocation problem in OMAS. This section covers the current status of the framework, while limitations and future enhancements are discussed in Section 8.3.

8.2.1 DTAACS-OK solution to the SAT-AP and MAT-AP

DTAACS-OK is a framework designed to address the task allocation problem in cooperative mission achievement in OMAS. In particular DTAACS-OK addresses the Single Agent Task Allocation Problem (SAT-AP) and the Multiple Agent Task Allocation Problem (MAT-AP) as defined in Chapter 2. The user expectations in regards to adaptability, autonomy, robustness, and security demand that the solution enable the agents in the organization to make their own decisions, react to changes in the environment and optimize the communication among them. The allocation of tasks is based on identical organizational knowledge being possessed by all agents in the organization when processing the same event. The organization knowledge is updated by exchanging messages among agents. This organization knowledge update occurs, in one way or another, in all solutions proposed in the litera-

ture. DTAACS-OK takes advantage of this organization knowledge and allocates the task avoiding the communication cost of auctioning tasks.

In DTAACS-OK, the agents in the system work cooperatively to achieve a mission. The DTAACS-OK framework is designed to handle complex missions represented by interrelated tasks. The mission is represented by an acyclic rooted tree where each node represents a task. Besides the parent-child relationships among tree nodes, DTAACS-OK considers other constraint relationships: *conjunctive*, *disjunctive*, *precedes*, *triggers*, and *negative-triggers*, which are defined in detail in Chapter 2.

In a cooperative environment, agents usually share information to keep each other aware of the status of the mission. One of the main features in DTAACS-OK is taking advantage of the information shared among the agents in a system. The framework includes by four main components: 1) *Distributed Transaction Component*, 2) *Distributed Organization Knowledge Component*, 3) *Distributed Task Allocation Component*, and 4) *Local Agent's Information Component*. These four components interact and provide the reasoning to find an efficient solution to the task allocation problem in agent organizations operating in dynamic and challenging environments.

DTAACS-OK keeps the relevant information in the Distributed Organization Knowledge Component (DOK), which is updated by the Distributed Transaction Component (DTC). The main purpose of the DTC is to ensure the *one-copy serializability property* of the organization knowledge. The DOK passes the organization knowledge to the Distributed Task Allocation Component (DTAC) to generate the assignments. DTAACS-OK also includes a Local Agent's Information Component, which stores the current agent's information that is used as part of the organization knowledge update.

Distributed Transaction Component. As previously mentioned, the main goal of the DTC is to ensure the one-copy serializability of the organization knowledge. In order to ensure this property, the DTC coordinates each transaction that occurs in the system

as explained below. The DTC is composed of two modules: the Transaction Generator Module, which receives events (See Table 4.1) from the agent while executing a task. Once it receives an event, the DTC proceeds to generate a new transaction, which is passed to the Transaction Manager. When the Transaction Manager receives a transaction, it establishes communication with the agents in the organization following a distributed transaction commitment protocol. If more than one agent wants to commit a transaction, a protocol to determine which agent to proceed is executed. When the agents agree to commit a transaction, each agent passes the committed transaction to the DOK. This guarantees the state of the DOK to be the same in all agents, and thus, the decision made after the commitment will be the same in each agent.

Distributed Organization Knowledge Component. The DOK stores the organization knowledge that is used when an assignment needs to be made. The DOK is consist of two modules.The first module is the Task Set Selection Module, which stores the task tree that represents the mission, the tasks ready to be assigned (Active Task Set), the tasks already achieved (Achieved Tasks Set), the tasks that are no longer needed to be pursued (Removed Task Set), and the tasks for which execution failed (Failed Task Set). These sets are updated by the committed transactions listed in Table 4.2. The second module in the DOK is the Organization Information Module that consists of two sets. The first set is the Agent Set that stores information about all the agents in the organization as shown in Table 4.3. The second set is the Capability Set that stores information about all the capabilities the agents contribute to the organization.

Distributed Task Allocation Component. The third component in DTAACS-OK is the Distributed Task Allocation Component (DTAC), whose main goal is to generate a new assignment set using the latest organization knowledge. This component integrates four main modules: (1) Allocation Algorithm, (2) Utility Function, (3) Utility Criteria Entity, and (4) Allocation Policies. The utility function is application dependent, and the assignment policies can be defined based on the optimization objectives in the system. The

allocation algorithms attempt to allocate the task in the mission until it is achieved, or a situation prevents the mission achievement.

Local Agent's Information Component. The fourth component in DTAACS-OK is the Agent's Local Information Component. This component stores the current status information of the agent, which may include physical characteristics of the robot, and its geographical location. Table 4.6 list some agent's information examples.

DTAACS-OK has been evaluated in different application domains in a simulation environment. The experiments were designed to evaluate DTAACS-OK in relation to communication cost and how the framework performs when the number of tasks increases. The experiments were run under scenarios where the communication among agents and agents' capability to achieve a task were stressed. Also, the DEMiRF-CF framework, which also addresses task allocation problem using a market based approach, was implemented in a Collaborative Assembling Object (CAO) application in order to compare communication costs.

The first application into which DTAACS-OK was integrated was HuRT-IED. This application simulates the mission of clearing an area of IEDs. In this type of application, it is beneficial to reduce the number of message agents exchange among themselves, an to react to messages that are lost and to agents failing to achieve a task.

The second application into which DETAACS-OK was integrated was the COA. This application simulates the mission of agents collaborating to assemble an object. The object is composed of different types of parts that must be assemble in a specific order, with some parts requiring more than one agent to be transported.

The results of these experiments are discussed in Section 8.4.

8.3 Future Work

In this section, some ways to improve DTAACS-OK are discussed. These potential improvements were selected based on the focus of this research: The main goal of this research was to offer a solution to the task allocation problem that supports system availability, reliability, robustness, elimination of single points of failure, and optimization of communication cost.

8.3.1 Identical Organization Knowledge

DTAACS-OK solves the problem of allocating tasks in OMAS in a distributed manner. The framework relies on each agent in the organization possessing identical knowledge at the time of making the same assignment. An important problem in distributed systems is the *network partition problem*. Protocols that guarantees one copy serialibility can handle network partition, but to reduce the inevitable effects on DTAACS-OK's efficiency due to the network partition problem, the concept of *agent neighborhoods* can be explored. Since the mission and all possible type of tasks and their required capabilities are known, agents in the organization can form *neighborhoods* based on the required and possessed capabilities. This neighborhood concept may reduce the number of agents that each agent needs to communicate with, thus lowering the risk of network partition. Also, a neighborhood representative can be elected to interact with other neighborhoods when needed. Since each agent possesses all needed information from the agents in the neighborhood, a triangulation can be used to update the required information in case of a network partition.

8.3.2 Coalitions

Some algorithms that generate coalitions have a high computational complexity, and it is information that is currently *redundantly* generated in DTAACS-OK. A possible way to reduce the overall degree of the polynomial complexity of DTAACS-OK algorithms may be to generate the possible coalitions in advance, and only update the information of these

coalitions at run time. Seeking for the most suitable coalition for a specific task using pre-generated coalition sets should reduce the computational complexity of DTAACS-OK algorithms.

8.4 Conclusion

In this research, DTAACS-OK has been designed and implemented to solve the task allocation problem in OMAS. Several performance test were conducted, integrating DTAACS-OK into applications in the exploration and collaboration domains. The results of the different experiments show that DTAACS-OK is an efficient, scalable and robust distributed framework for OMAS that solves the task allocation problem, while maintaining a low communication cost.

In the HuRT-IED is an application, DTAACS-OK was integrated to evaluate the framework in regards of communication cost and agent task failures to achieve tasks. The results of these test show that DTAACS-OK is a low communication cost solution to the task allocation problem, that increases the communication security by lowering the number of messages exposed. The results also illustrate that DTAACS-OK efficiently reallocate tasks when an agent is not able to achieve it.

In the COA application, DTAACS-OK was integrated to evaluate the framework in regards of communication cost and task reallocation when agents failed to achieve tasks. In this particular application, a task could require more than one agent so a coalition needs to be formed, and collaboration among the agents forming the coalition occurs. The result of these experiments show that DTAACS-OK is a low communication cost for the task allocation problem. Part of the reduction in communication cost is due to the low communication among the coalition's agents regarding the coalition formation and termination.

DTAACS-OK versus DEMiRF-CF

When analyzing computational and communication complexity of DTAACS-OK and DEMiRF-CF, both belong to the polynomial category. However when both frameworks were integrated to the COA application, the results show that DTAACS-OK offers an important reduction on the number of messages exchanged among agents. DEMiRF-CF uses a marked-based approach while DTAACS-OK uses an approach maintaining identical organization knowledge in all agents. In DTAACS-OK, messages about mission execution status are exchanged between agents in the organization, which is common in most solutions, including DEMiRF-CF. In DTAACS-OK these mission execution update messages are used to update the agent's organization knowledge. The extra payload incurred in DTAACS-OK to support maintaining identical organization knowledge is determined by the protocols used to provide one-copy serializability property to the organization knowledge. These protocols run in polynomial message complexity, which does not increase the complexity.

8.5 Summary

In this chapter, a brief summary of the research conducted in this dissertation was presented. In Section 8 a brief definition of the task allocation problem is presented. Section 8.1 presents a summary of some relevant approaches that also tackle the task allocation problem. Section 8.2 presents a summary of DTAACS-OK, the framework proposed in this research to address the SAT-AP and MAT-AP in cooperative OMAS. Finally, in Section **??** the limitations of the DTAAACS-OK as the research in this dissertation was concluded and the future work is discussed.

Bibliography

[1] The contract net protocol. *IEEE Transactions on Computers*, page 1105, December 1980.

[2] Amr El Abbadi, Dale Skeen, and Flaviu Cristian. An efficient, fault-tolerant protocol for replicated data management. *PODBS*, pages 215–229, 1985.

[3] Sherief Abdallah and Victor R. Lesser. Modeling task allocation using a decision theoretic model. In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors, *AAMAS*, pages 719–726. ACM, 2005.

[4] Greg Gagne Abraham Silberschatz, Peter Baer Galvin. *Operating System Concepts*. Jhon Wiley and Sons, Inc., New Jersy, USA, 2005.

[5] Patrick Beautement, David N. Allsopp, Mark Greaves, Steve Goldsmith, Shannon Spires, Simon G. Thompson, and Helge Janicke. Autonomous agents and multi -agent systems (aamas) for the military - issues and challenges. In *Defence Applications of Multi-Agent Systems*, pages 1–13, 2005.

[6] Sally C. Brailsford, Chris N. Potts, and Barbara M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119(3):557–581, December 1999.

[7] L. Brickman. *Mathematical introduction to linear programming and game theory*. Undergraduate texts in mathematics. Springer-Verlag, 1989.

[8] Peter Brucker. Scheduling and constraint propagation. *Discrete Applied Mathematics*, 123(1-3):227–256, 2002.

[9] Thomas L. Casavant and Jon G Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEESE*, 14(2), February 1988.

[10] S. Ceri, M. Houtsma, A. Keler, and P. Samarati. A classification of update methods for replicated databases. Technical Report CS-TR-91-1392, Stanford University, May 1994.

[11] Georgios Chalkiadakis and Craig Boutilier. Sequential decision making in repeated coalition formation under uncertainty. In Lin Padgham, David C. Parkes, Jörg P. Müller, and Simon Parsons, editors, *AAMAS (1)*, pages 347–354. IFAAMAS, 2008.

[12] Archie Chapman, Rosa Anna Micillo, Ramachandra Kota, and Nick Jennings. Decentralised dynamic task allocation: A practical game?theoretic approach. In *The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09)*, pages 915–922, May 2009.

[13] V. Chvatal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[15] Silvia Silva da Costa Botelho and Rachid Alami. M+: A scheme for multi-robot cooperation through negotiated task allocation and achievement. In *ICRA*, page 1234, 1999.

[16] Silvia Silva da Costa Botelho and Rachid Alami. M+: A scheme for multi-robot cooperation through negotiated task allocation and achievement. In *ICRA*, page 1234, 1999.

[17] S.A. DeLoach and M. Miller. A goal model for adaptive complex systems. *International Journal of Computational Intelligence: Theory and Practice*, 16(1):5(2), 2010.

[18] Scott A. DeLoach, Walamitien H. Oyenan, and Eric T. Matson. A capabilities-based model for adaptive organizations. *Autonomous Agents and Multi-Agent Systems*, 16(1):13–56, 2008.

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

[20] Sally M. El-Ghamrawy, Ali I. El-Desouky, and Ahmed I. Saleh. Distributed task allocation in multi-agent system based on decision support module. 2010.

[21] J. Ferber. *Multi-agent Systems: Introduction to Distributed Artificial Intelligence*. Addison Wesley, 1999.

[22] Donald Francis Ferguson. The application of microeconomics to the design of resource allocation and control algorithms, 1989.

[23] L. R. Foulds. *Combinatorial Optimization for Undergraduates*. Springer Verlag, New York, 1984.

[24] L.R. Foulds. *Combinatorial optimization for undergraduates*. Undergraduate texts in mathematics. Springer-Verlag, 1984.

[25] Thomas L.C. French, S. and White D.J. *Operational research techniques*. E. Arnold, 1986.

[26] Vijay K. Garg, Ph.D. *Elements of distributed computing*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[27] Brian P. Gerkey and Maja J. Mataric. Multi-robot task allocation: analyzing the complexity and optimality of key architectures. In *ICRA*, pages 3862–3868. IEEE, 2003.

[28] Brian P. Gerkey and Maja J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *I. J. Robotic Res*, 23(9):939–954, 2004.

[29] D. M. Gordon. interaction patterns and task allocation in ant colonies. *Information Processing in Social Insects, C. Detrain, J. L. Deneubourg, and J. M. Pasteels, Eds. Basel, Switzerland: Birkhauser Verlag*, page 5167, 1999.

[30] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2000):277–296, 2000.

[31] url = http://macr.cis.ksu.edu/cros Kansas State University MACR Lab, title = CROS: The Cooperative Robotic Organization Simulator.

[32] Jane W. S. Liu. *Real-Time Systems*. Prentice Hall, New Jersy, USA, 2000.

[33] Thomas W. Malone. What is coordination theory? In *Proceedings of the National Science Foundation Coordination Theory Workshop*, February 1988.

[34] Multiagent and Cooperative Reasoning Laboratory. Human - robot teams @ONLINE, 2011.

[35] L. E. Parker. ALLIANCE: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 23 1998. Publisher: IEEE Robotics and Automation Society.

[36] L. E. Parker and F. Tang. Building multirobot coalitions through automated task solution synthesis. In *Proceedings of the IEEE, Vol. 94, No. 7. (21 2006),*, July 2006.

[37] K. Ramamritham and J. A. Stankovic. Dynamic task scheduling in distributed hard real-time systems. In *Proceedings of the 4th International Conference on Distributed Computing Systems*, pages 96–107, 1984.

[38] I. Rhee. Optimal fault-tolerant resource allocation in distributed systems. *(Proc. 7th IEEE Symp. on) Parallel and Distributed Processing*, pages 460–469, 1995.

[39] L. C. Thomas S. French, R. Hartley and D.J. White. *Operational Research Techniques*. Edward Arnold, 1986.

[40] Sariel Sanem and Balch Tucker. *A Distributed Multi-Robot Cooperation Framework for Real Time Task Achievement*, chapter 19, pages 187–196. Springer Verlag, 2006. http://www2.itu.edu.tr/šariel/publications.php.

[41] Sanem Sariel. *An Integrated Planning, Scheduling and Execution Framework for Multi-Robot Cooperation and Coordination*. PhD thesis, Istanbul Technical University, Institute of Science and Technology, Istanbul, Turkey, June 2007.

[42] Sanem Sariel, Tucker Balch, and Nadia Erdoğan. Incremental multi-robot task selection for resource constrained and interrelated tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2314–2319, 2007. http://www2.itu.edu.tr/šariel/publications/IROS_Sariel_2007.pdf.

[43] A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1998.

[44] Jonathan R. Senning. Operating systems and computer architecture, 2000.

[45] Onn Shehory and Sarit Kraus. Task allocation via coalition formation among autonomous agents. In *IJCAI (1)*, pages 655–661, 1995.

[46] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, May 1998.

[47] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[48] Peter Stone and Manuela M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Auton. Robots*, 8(3), 2000.

[49] F. Tang and L. E. Parker. Layering coalition formation with task allocation. In *The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09)*, May 2006.

[50] Johannes van der Horst and Jason Noble. Distributed and centralized task allocation: When and where to use them. In *SASO Workshops*, pages 1–8. IEEE Computer Society, 2010.

[51] L. Vig and J. A. Adams. A Framework for Multi-Robot Coalition Formation. In Bhanu Prasad, editor, *IICAI 2005*, Pune, India, December 2005.

[52] Lovekesh Vig and Julie Adams. Issues in multi-robot coalition formation. In Lynne Parker, Frank Schneider, and Alan Schultz, editors, *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pages 15–26. Springer Netherlands, 2005.

[53] Michael Wilkes. *Operational Research: Analysis and Applications*. McGraw-Hill, 1989.

[54] Michael Woolridge and Michael J. Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[55] J. Wu. *Distributed System Design*. CRC Press, 1999.

[56] K. Zhang, E. Collins, and D. Shi. Centralized and distributed task allocation in multi-robot teams via a stochastic clustering auction. *ACM Transactions on Autonomous and Adaptive Systems (to appear)*.

[57] Robert Zlot and Anthony Stentz. Market-based multirobot coordination for complex tasks. *I. J. Robotic Res*, 25(1):73–101, 2006.