INTELLIGENT DECISION SUPPORT FOR MARINE SAFETY AND SECURITY OPERATION CENTRES

by

Ali Khalili-Araghi

B.Sc., University of Tehran, 2007

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Summer 2011

APPROVAL

Name:	Ali Khalili-Araghi
Degree:	Master of Science
Title of thesis:	Intelligent Decision Support for Marine Safety and Security Operation Centres

Examining Committee: Dr. Steven Pearce
Chair

Dr. Uwe Glässer, Professor
Senior Supervisor

Dr. Alexandra Fedorova, Assistant Professor
Supervisor

Dr. Lou Hafer, Professor,
SFU Examiner

Date Approved: August 24, 2011

Abstract

Surveillance of large volume traffic demands robust and scalable network architectures for distributed information fusion. Operating in an uncertain environment and the ability to flexibly adapt to dynamic changes in resource availabilities are critical for the success of surveillance and rescue missions. In this thesis the architectural design of a decision support system for a Marine Safety and Security Operation Centre (MSOC) is presented. The goal of this system is to improve coordination in emergency response services. The system design emphasizes robustness and scalability through its decentralized control structure, automated planning, dynamic resource configuration management and task execution management under uncertainty. The proposed model is described in abstract functional and operational terms based on the Abstract State Machine (ASM) paradigm and the CoreASM open source tool environment for modeling dynamic properties of the system. An example scenario from the marine operations domain is described in detail, and afterwards, an experimental analysis which evaluates the validity of our system under various scenarios are presented at the end.

Keywords: Decision Support Systems; Multi-agent Systems; Marine Safety and Security; Abstract State Machines; CoreASM; Distributed Systems; Automated Planning

To my beloved grandparents...

“Two roads diverged in a wood, and I..
I took the one less traveled by,
And that has made all the difference.”

— *The Road Not Taken*, Robert Frost, *1920*

Acknowledgments

First and foremost, I would like to offer my sincerest gratitude to my supervisor, Dr. Uwe Glässer, who supported me throughout this thesis with his patience and knowledge whilst allowing me the room to work in my own way. Without his guidance, I could never have reached the heights or explored the depths; one simply could not wish for a better or friendlier supervisor.

I also wish to express my especial appreciation to Roozbeh Farahbod, who was not only my mentor but also a close friend who taught me innumerable lessons and gave me valuable insight into research. It is not that often that one finds a colleague who always finds time to offer help.

Last, but not least, I would like to express my heartfelt thanks to my dear one, Azin Dastpak, who stood by me through it all and gave me the utmost encouragement and support.

Contents

Approval **ii**

Abstract **iii**

Dedication **iv**

Quotation **v**

Acknowledgments **vi**

1 Introduction **1**

1.1 Motivation 1

1.2 Objectives 5

1.3 Significance 6

1.4 Thesis Organization 7

2 Background **8**

2.1 Command and Control Hierarchy 8

2.2 Cooperative Models 9

2.2.1 Definitions 10

2.2.2 Collaborative Systems 10

2.2.3 Swarms 11

2.2.4 Coalition Systems 12

2.2.5 Clusters 14

2.3 Conclusions 15

3 Problem Characteristics 17
3.1 System Distribution . . . 18
3.2 Dynamic Environment . . . 19
3.3 Uncertainty . . . 20

4 Proposed Model 22
4.1 High Level View . . . 22
4.2 System Architecture . . . 25
4.2.1 Planning . . . 27
4.2.2 Resource Management . . . 28
4.2.3 Execution Management . . . 29
4.2.4 Tasking . . . 29
4.2.5 Decentralized Control . . . 30
4.3 Reference Model . . . 30
4.3.1 An Introduction to User Requirements Notation . . . 30
4.3.2 Describing the Abstract Generic Scenario . . . 31

5 Formal Model 33
5.1 Basic Concepts . . . 34
5.1.1 Node . . . 34
5.1.2 Capability . . . 34
5.1.3 Layered Architecture . . . 37
5.1.4 Capability Pattern . . . 38
5.2 Task Management . . . 40
5.2.1 Node Program . . . 41
5.2.2 Task Lifecycle . . . 42
5.2.3 Matching Child Node . . . 47
5.3 Simple Scenario . . . 51

6 Experimental Analysis 54
6.1 Evaluation metrics . . . 55
6.2 Test Setup . . . 57
6.3 Simulation Results . . . 58
6.4 Conclusions . . . 60

7 Conclusions **62**

A ASM Formalism **64**

Bibliography **67**

Glossary **73**

List of Figures

1.1 Maritime traffic in Strait of Dover 2
1.2 Density of marine traffic at BC Coast area 3

2.1 Command and control hierarchy 9
2.2 Swarms robots 12
2.3 Coalition organization 13
2.4 Hierarchical clustering 15

3.1 Problem Characteristics 20

4.1 Nodes represent physical or logical resources. 24
4.2 Components of the System Architecture 27
4.3 Hierarchical Task Network (HTN) Example 28
4.4 Abstract Generic Scenario of the NADIF System 31

5.1 Capability Hierarchy example 35
5.2 RMA is organized in four layers. 37
5.3 Control state diagram of a Search-and-Rescue task 40
5.4 Activity cycle of a node 41
5.5 Task Lifecycle 42
5.6 Control state diagram of matching tasks to child nodes 48
5.7 Structure of the resource hierarchy 53

6.1 Resource network for scenario testing 57
6.2 Results of scenarios runs 59
6.3 Receiver Operating Characteristic diagram 60

List of Tables

6.1 Types of test results 56

Chapter 1

Introduction

This thesis focuses on the architectural design of a surveillance system to be used for marine security centres. A comprehensive model is proposed, which captures all the requirements needed for surveillance operations. Such a model encompasses different features of a multi-agent architecture to address the surveillance problem by providing a robust and scalable system which can be used in emergency response situations. Moreover, the model is precisely formalized using the Abstract State Machine paradigm, which provides abstract specifications of the system. Furthermore, by using CoreASM , the ASM descriptions can be executed for simple scenarios to validate the model.

1.1 Motivation

In recent years, the increasing international trade has led to more cargo, more ships, and more traffic in ports, coastal areas and regional waterways. Figure 1.1 shows the Strait of Dover (located between France and England).[1] In the map, colored icons represent vessels in water. Each type of vessel is represented by a unique icon shape. As depicted in the figure, there's a huge vessel traffic in this region, which is the busiest international seaway in the world with over 400 commercial vessels passing through daily[2] Such tremendous increase in maritime traffic means considerable challenges for many countries.

Every waterway introduces unique safety and security challenges. The more dense the

[1]Provided by www.marinetraffic.com

[2]Reported by Department of Transport in UK `http://www.dft.gov.uk`

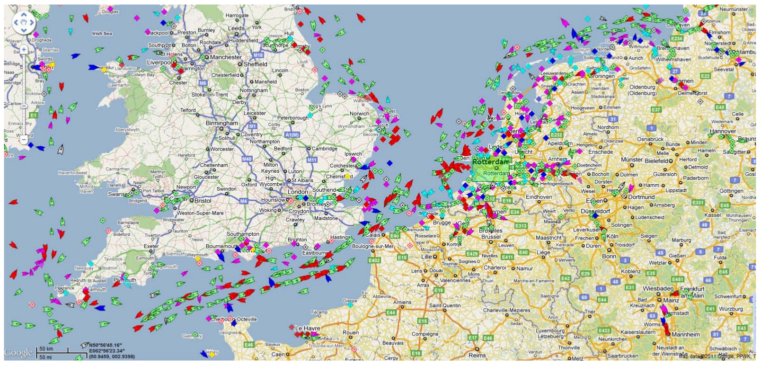


Figure 1.1: Maritime traffic in Strait of Dover

area, the more challenging the issues are. Ports and coastal regions are recognized as the most dense locations in the world with large volume of maritime traffic. For instance, Figure 1.2 illustrates the density of vessel traffic in BC coast area. The colors represent the volume of traffic; colors towards red show more traffic density. As depicted in the picture, the majority of vessel traffic in the BC coast is in the south, through the Strait of Georgia, which is a narrow waterway between Seattle port and Vancouver port with a great number of inbound and outbound marine traffic. Such huge traffic in this region introduces many challenges for the United States and Canada. For instance, in emergency response situations or dealing with national security issues such as piracy, smuggling, and illegal immigration.

In general, Canada, with the longest coastline in the world (243,042 km), faces the problem of large volume coastline surveillance [1]. Here, an automated tracking system such as AIS (Automatic Identification System) has greatly improved the monitoring of maritime traffic. AIS technology is used on ships and by Vessel Traffic Services[3] to provide information needed for identifying and locating vessels in the water. This can be done by electronically exchanging data with other nearby ships and VTS stations. AIS has revolutionized vessel traffic control and marine navigation collision avoidance. Still, managing the deployment of resources in a rapidly changing environment is an issue that calls for a *Decision Support System* to mitigate the problem.

According to the definition by Turban and Aronson [23], the central purpose of a Decision

[3]A Vessel Traffic Service (VTS) is a monitoring system designed for marine traffic; the system is established by competent authority to keep track of vessel movements and to provide a safe navigation in the relevant area [4, 49]. This system is very similar to air traffic control designed for aircrafts.

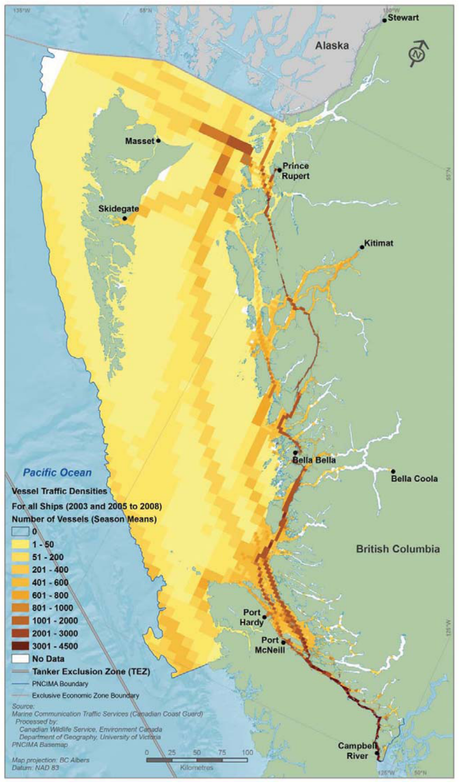


Figure 1.2: Density of marine traffic at BC Coast area[4]

Support System (DSS) is to support and improve decision making. Such systems have been applied in many different domains. For instance, they have been used to address the gate allocation problem in international airports [18] and in railway scheduling problems for the Canadian Pacific Railway [51].

One of the main critical fields in the decision making process is *Situation Awareness* (SA). According to Endsley's definition [25], "Situation Awareness is the perception of the elements in the environment, comprehension of their meanings, and projection of their status in the near future". If SA is considered as a state of knowledge, the process that leads to achieve this state is called *Situation Assessment*. Therefore, in order to have a consistent understanding of the elements in the system, situation analysis of the current state is necessary.

[4]Provided by the Pacific North Coast Integrated Management Area Initiative. 2011. Atlas of the Pacific North Coast Integrated Management Area. Available at `www.pncima.org`

The design of decision support systems has been shifted towards a more distributed and collaborative paradigm. Agent-based technologies have greatly enhanced the capabilities of decision support systems [73] . There are many definitions for an "agent" in the literature. Wooldrige et al. [76] distinguished two notions for the term agent, weak and strong. In weak notion, an agent is defined as a distinct, autonomous, and active entity with features including pro-activeness, reactivity and social ability (the capability to interact with other agents or humans). In strong notion, Intelligent Agents (IA) have other extensive features such as human-like characteristics, e.g., knowledge, beliefs, desires and intentions. In Multi-agent systems, cooperation among agents refers to situations in which several agents in the system are trying to fulfill a shared goal through different means of communication. From this perspective, a maritime surveillance system can be considered as a cooperative multi-agent system. For this research, cooperative multi-agent systems have been thoroughly explored in diverse areas ranging from biology[5] to engineering applications[6]. Some examples will be described in more detail in Chapter 2.

In this thesis, an intelligent decision support system, called NADIF (Net-Enabled Adaptive Distributed Information Fusion), is proposed for large volume coastal surveillance [7]. The system is based upon a realistic application scenario and have been designed over the ideas used from the design and development of the CanCoastWatch system [72, 75]. Moreover by using the *Abstract State Machine* (ASM) formalism, the ideas have been implemented at an abstract level of understanding [30, 31]. Furthermore, promising evaluation results show that the proposed architecture is indeed a good choice for such a surveillance system [43, 42]

Our model consists of three major building blocks: 1) *Information Fusion Engine*, 2) *Decision Support Engine*, 3) *Resource Management Engine*.

Information fusion helps us to obtain useful information about the current state of the system. The raw data, indicating the status of each resource, are dynamically collected using the updated information within the system. Subsequently, more useful knowledge (i.e. Situation Awareness) can be generated based on these information by different means of analysis techniques, called Situation Assessment. The knowledge can then be used as feedback to the system or can be reported back to the commander [16].

Decision support is one of the main aspects of a surveillance system. Most of the time, there are different ways to make a decision. For instance, finding a proper resource for a

[5]such as Swarms [54] and ant colonies [22]

[6]for instance, intelligent vehicle highway system [44] and cooperative control for UAVs [24]

"rescue" mission can be difficult when all of the resources have already been assigned to other tasks. This problem can be overcome by choosing one or more low-priority tasks and suspending them, so that their resources can be released and assigned to another task. In general, task prioritizing can be beneficial in critical situations with limited resources. Therefore, a *decision support engine* is needed in a complicated system which deals with a great number of resources and potentially conflicting tasks, hence, different choices for any decision.

A surveillance system may have to deal with a number of distributed heterogeneous resources which are autonomously interacting with each other by different means of communication. Therefore, the role of a *resource management engine* in monitoring and management of such many resources is crucial. In addition, by having a resource management engine, more complex resource configuration structures (e.g. clusters, trees, and etc.) can be exploited which would then improve the performance of task assignments by use of new features such as capability aggregations, load balancing, and so on. Lastly, the surveillance system works under dynamically changing conditions. The environment can influence the availability and performance of the resources unpredictably. For this reason, having the status of the resources updated at all time is essential for the performance of other engines, such as task management, in order to accomplish the missions. The focus of this thesis is on the design of a collaborative decision support system with an underlying configuration management model. Moreover, information about the current state of the system is gathered dynamically in order to keep a consistent view of the world.

1.2 Objectives

The goal of our work is to provide the architecture and core mechanisms of a decision support system in the domain of maritime operations related to safety and security. The system should configure and manage the resources at run-time while facilitating the decision making process for humans in emergency response situations and for preventing smuggling operations, especially near ports and harbors. The surveillance system has some main characteristics that need to be addressed:

- *Dynamic environment*: unpredictable changes in the environment or the availability of resources in the system;

- *System Distribution*: people and equipment are working in different locations in real-time;
- *Uncertainty*: lack of perfect and complete knowledge of the environment or unexpected results of actions performed by the system.

In this regard, the proposed model benefits from the use of a multi-agent system design approach, which can also increase the robustness and scalability of the system. In addition, by using ASM, the specification of the system is formalized in abstract descriptions. By further refining of the model, it can be made executable. To achieve this goal, an ASM *ground model* of the system [11] is formally described, which can be further enriched with more detail through stepwise refinements. Basically, a ground model is a precise yet abstract description of the system. It is also flexible for future changes. Such a model is abstract yet complete insofar as completeness refers to the presence of every semantically relevant feature in the system [10].

1.3 Significance

According to the statistics [37], software project failures cost billions of dollars annually. Thus, it is crucial to identify any defect in early stages of software development, to reduce the possibility of failure. As mentioned before, providing safety and security for maritime traffic is a great national challenge that calls for an intelligent decision support system. Therefore, the complexity and importance of the system, accentuates the need for validation of the model from the beginning of design process. As the earlier a defect is found, the cheaper it is to fix it.

When tackling complex systems, it is difficult, if not impossible, to test the correctness of the design model before implementation. In this work, we use ASM methodology, to formally describe the specifications of the model. ASM allows us to have models with different levels of abstraction. Starting from the ground model, a hierarchy of intermediate models can be obtained through stepwise refinements. The uniqueness of this approach lies in the fact that with use of ASM, the system model can be tested for validation, in each level of abstraction, before actually building the system.

1.4 Thesis Organization

The thesis is organized as follows: Chapter 2 provides an overview of the related work. In this chapter, some similar architectures of cooperative multi-agent systems are explored and the *command and control* hierarchy, used in military operations, is introduced. In Chapter 3, the problem domain is described in more detail, with the focus being on the characteristics of the system that we are dealing with.

Chapter 4 introduces our proposed model for the system. An overview of the architecture and main components of the model is presented in this chapter, while explaining the functionalities of each component with use of *User Requirement Notation*. Afterwards, in Chapter 5, formal specifications of the model is presented by use of the ASM paradigm. In addition, the system is implemented in parts using CoreASM framework to simulate a simple scenario in marine operations domain, which is explained at the end of the chapter. Chapter 6 will evaluate the system runs over a set of scenarios. In this chapter robustness of the system is measured in dynamic changing environment. Finally, Chapter 7 concludes the thesis by presenting the significance of this research and introducing the subjects of future work.

Chapter 2

Background

This chapter briefly discusses background concepts and related works relevant to surveillance systems along with domain-specific aspects presented in the literature. We employ such ideas to formulate a model that accurately captures the characteristics, functionalities and requirements of the target system.

2.1 Command and Control Hierarchy

One of the essential concepts in dealing with surveillance systems is *command and control* (C2), which is mostly used in military domains [57]. As defined by the Department of Defense in the Dictionary of Military and Associated Terms [70]:

> *Command and control*: "The exercise of authority and direction by a properly designated commander over assigned and attached forces in the accomplishment of the mission. Command and control functions are performed through an arrangement of personnel, equipment, communications, facilities, and procedures employed by a commander in planning, directing, coordinating, and controlling forces and operations in the accomplishment of the mission."

By this definition, missions are assigned by the commander to subordinate personnel; any failure or completion of the assigned tasks should be reported back to the commander for further assignments. This model works very well when there is a small number of personnel involved. It is not very difficult to imagine the degree of complexity in a coastal surveillance system with a large number of widely spread out heterogeneous resources and personnel.

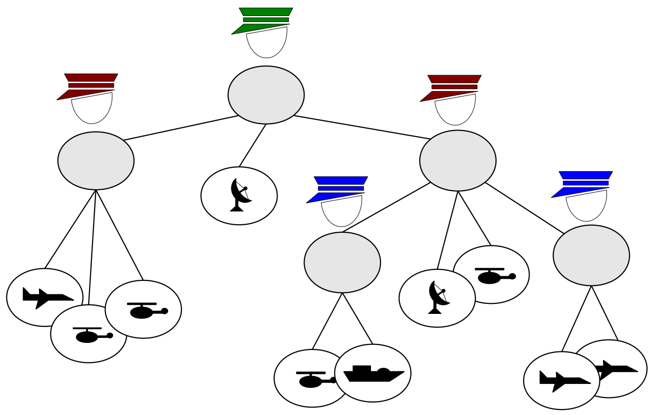


Figure 2.1: Command and control hierarchy

Hierarchical layering is one of the general approaches for managing complex systems. It has also been used by the military for effectiveness and efficiency of command and control structure. In a system comprises of multiple nodes, organized in a hierarchical structure, the scope of higher level nodes are broader and the details are skipped while the lower level nodes are merely responsible for certain tasks including specific details [3]. Using hierarchical structuring will be helpful in terms of decreasing the complexity of the system. This is due to the fact that a node at a certain level of the hierarchy does not have to deal with broad scopes and low-level details at the same time.

For example, at the top level of the command and control hierarchy shown in Figure 2.1, new tasks are selected and introduced to the system by top commander based on high-level goals and priorities, whereas the subordinates deal with the details of task execution.

2.2 Cooperative Models

Cooperative Multi-Agent System is a well-studied concept in the area of multi-agent systems. There are also different architectural models for such systems representing the nature of cooperation between the agents. For this thesis, different architecture models have been studied, which will be introduced shortly, and the appropriate model for a marine safety and security system is proposed in next chapters.

2.2.1 Definitions

According to definitions by Pannit [62], an agent can be described as a high level autonomous computational mechanism. Its actions are based on the information (sensors, feedback) received from the environment. A multi-agent environment is an environment, where each agent interacts with more than one agent. There may be other constraints on such environments in terms of distribution of information within the system. For instance, a given agent might not be aware of the status of other agents or their internal states due to some restrictions in the system.

As Franklin and Graesser explained [36], in *cooperative* multi-agent systems, the agendas of the agents consist of cooperation with other agents in the system. Such systems are divided into two types: Communicative and Non-communicative. Communicative is when the agents are cooperating with each other by different means of communication, while in non-communicative systems, there is no communication between the cooperative agents (agents can perform cooperations by monitoring the behaviour of other agents and responding accordingly). In another perspective, Doran [21] describes cooperation as a property of the actions of agents. In this view, agents either have shared goals in common or they perform actions that help the other ones to achieve their goals.

Based on the definitions provided above, the coastline surveillance system in this study can be a good example of a cooperative multi-agent system, where there are different agents (resources) cooperating with each other to fulfill introduced tasks.

In general, there are various types of cooperative systems. Each has its own characteristics and architectural model. Some of such systems that are related to our proposed model have been studied and are briefly introduced in this chapter. Understanding the architectural models of these systems is beneficial for our proposed model of the coastline surveillance system which will be described in more detail in Chapter 4.

2.2.2 Collaborative Systems

In a collaborative system, a shared task is performed by multiple agents, and are usually connected to each other from different locations. The main feature that distinguishes such systems from a larger domain of distributed applications, is the fact that the agents are collaborating to achieve common goals. They also need to closely interact with each other by exchanging requests and sharing information or status among each other [20].

Chat system can be a good example of a collaborative system as all the users involved in the chat session need to collaborate so that no comments are missed. However, in the email system the only issue that the user is concerned about is the delivery of the message to the right person. Therefore, there is no intention to have any collaboration in such systems. Some other examples of collaborative systems are: shared white-boards, interactive chat, and coordinated data search agents (e.g., web ”robots”) [35].

There are some issues with collaborative systems. These issues are related to the concurrency of the actions happening in the system or the collaboration among the agents. Some of these issues, categorized by Farely in [35], are briefly explained here.

- *Communication needs*: Since in collaborative systems various agents are collaborating with each other dynamically, the system should be able to support different types of message passing among agents. The routing mechanism can be point-to-point, broadcast or narrowcast based on the application’s needs. For example, in a chat system a user can send a message to the entire users in the specific chat room or only to another agent as a private message.

- *Maintaining agent identities*: In a collaborative system each agent should be recognized with a unique identity, which can be used in different applications such as message passing, task assignments, etc. In addition authentication of agents prior to accessing a shared resource can be facilitated by means of such unique identities.

- *Shared state information*: Generally, information is shared within many distributed systems. Such a feature is even more highlighted in collaborative systems where, intrinsically, cooperation can take place by sharing a set of data among different agents. In addition, concurrent updates on the shared state information can take place by various agent at the same time, hence preserving consistency and validity of such information is a challenging issue that needs to be carefully addressed.

2.2.3 Swarms

In Swarm systems, a large number of simple agents locally interact with each other and the environment without the presence of a central controller [59]. In such systems, agents do not necessarily have an understanding of the whole system, i.e. each agent has its own view of the world. The agents are self-organizing based on the emergent behaviour

Figure 2.2: Swarms robots

types of simple interactions. This behaviour is also observed in insects and flocks of birds. Bonabeau et al. [19], who studied self-organization in social insects, reveal that interactions among agents that show simple behaviours can result in complex collective behaviours in the system. The emergent behaviour is also referred to as the macroscopic behaviour whereas local interactions are referred to as the microscopic behaviour.

In this regard, the concept of self-organization can be seen easily in swarm systems. Generally, self-organization is defined as the emergence of global patterns in a system as a result of many low-level interactions which utilize only local information [19]. For instance, in Swarm Robotics (see Figure 2.2), a huge number of simple physical robots interact with each other and the environment. These communications provide feedback to the system, hence initiating emergent global behaviour in the system. Such behaviour is called *Swarm Intelligence* [9], which can be originated from highly simple rules for individual agents.

2.2.4 Coalition Systems

In many diverse areas, the concept of *coalition* has been well studied and based on the applications, the idea can be used to enrich cooperations in the system. It has also been proved that using coalition of individuals is a practical approach in multi-agent systems [48]. Generally speaking, working in groups can imply more utilities. This is the motivation behind co-ops, clubs, and unions. In computational domains coalitions can cause more

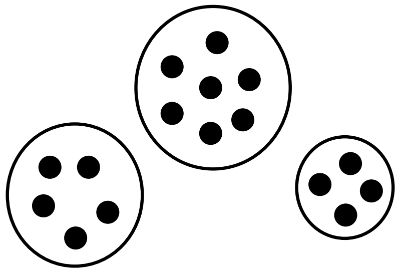


Figure 2.3: Coalition organization

efficient task allocations, or greater ability for solving problems than any single agent can offer [67]. Practically, in a multi-agent system, any subset of the agents can be a potential coalition. In addition as Horling and Lesser stated in [48]:

> "*Coalitions in general are goal-directed and short-lived; they are formed with a purpose in mind and dissolve when that need no longer exists, the coalition ceases to suit its designed purpose, or critical mass is lost as agents depart.*"

There are also various related studies that expand the coalition domain. In such extensions, coalition can be defined as longer-term agreements which are based on trust [17]. In addition, iterative formation of multiple coalitions can take place in an environment where tasks in the system are dynamic [58]. Although coalitions can be formed in both cooperative and self-interested populations, cooperative community is more related to the scope of this research.

Figure 2.3 shows a simple coalition structure in a population of agents. The organizational structure within a coalition is often flat. However, there might be a representative agent for each group [55, 53]. Such leading agents are in charge of interactions among the groups. For instance, if one coalition needs the results of another one, there might be an agreement upon the deadline between the two groups. This arrangement can only be done by the interactions between the two representatives.

In general, coalitions restrict the scope of interactions among agents while increasing the effectiveness of the whole system [68]. In addition, there is always a cost to form any coalition in the system. Therefore, coalition formation turns into a problem of choosing the appropriate subset of agents $S \subset$A to maximize the utility of the group that can be achieved in the environment. which the group can achieve in the environment. There are several algorithms for coalition formation in a multi-agent systems. Among those, Soh et al. [69] present a technique for coalition formation, where agents have incomplete and uncertain knowledge and must achieve their goals in real-time response situations.

2.2.5 Clusters

Classification goes back to early stages of human beings and can be considered as a basic ability of living creatures to group similar objects together [27]. In classification, the objects in a set are investigated to see if they can be placed into groups (or classes, or clusters), so that objects in a group are similar to each other and different from objects in other groups as stated by Gordan in [45].

Use of classification is widely spread in different areas such as biology, medicine, marketing, etc. With the growing number of large databases in many areas of science, it is essential to organize large data sets so that they can be more easily understood and information retrieved more efficiently.

There are different numerical techniques for deriving classifications. Conversely, the same procedure may be known by various names, depending on the application domain. For instance, *numerical taxonomy* in biology, *segmentation* in marketing, and *unsupervised pattern recognition* in artificial intelligence are all using the same concept of the classification. Nonetheless, *cluster analysis* refers to a method to uncover relevant groups in data sets, in which objects resemble each other in a cluster and are different in some respects from the objects in other clusters [27]. Although clusters do not have overlaps with each other in general, there are some applications where an element belongs to more than one cluster.

In this regard, there are a number of applications that need a hierarchically-nested set of partitions. For instance, in biological taxonomy, an object is commonly regarded as belonging successively to a species, a genus, a family, and an order. Secondly, hierarchical classification has been found to be a useful model in several topics in psychology and cognitive science, for example in the analysis of the phrase structure of sentences and the definition of a story grammar [6].

Although the mathematical definition of a hierarchical cluster is not presented in this thesis for the sake of simplicity, an example is provided to elaborate this concept and to help intuitively understand such structure. Suppose we have a set of objects $A = \{1, 2, ..., 10\}$. A hierarchical classification of this set can be represented as Figure 2.4. In such tree a leaf or terminal node represents an object of the set, and each internal node shows a non-singleton

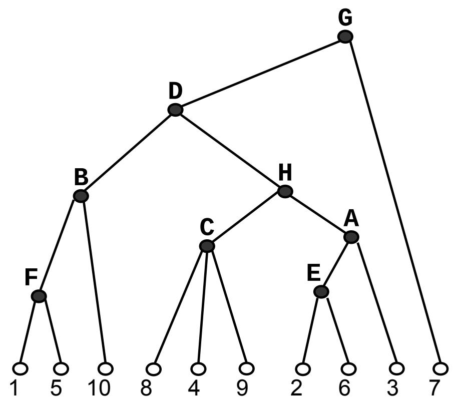


Figure 2.4: Hierarchical clustering

subset of objects. In this example the tree has the following subsets: {1, 5}, {2, 6}, {8, 4, 9}, {1, 5, 10}, {2, 6, 3}, {8, 4, 9, 2, 6, 3}, and {1, 5, 10, 8, 4, 9, 2, 6, 3} where each can be identified with internal nodes labeled *A–G*. Such a tree is called a *rooted tree* with labeled leaves, in which all the internal nodes have degree of greater than two (except the root, which its degree is equal to two). A binary tree is a sort of unordered rooted tree, in which all the internal nodes have degree three (except the root). There are many classification algorithms using binary trees.

There are also other mathematical concepts associated with the tree presentation of hierarchical clusterings (e.g. dendrograms, valued trees, etc.), which are beyond the scope of this research.

2.3 Conclusions

Considering the trade-off between the *behavioural simplicity* of the agents and the *structural simplicity* of the system, there is a spectrum of different architectural models available. In one end of the spectrum, we have swarm models where each agent has low intelligence and is capable of performing simple tasks. For such systems exploiting a huge number of simple agents can increase performance and robustness of the system. The latter is essential for a system working in a dynamically changing environment where unpredictable changes can affect the availability and performance of the agents. Nevertheless, a robust system is functional in spite of resource failures. In the other end of the spectrum there is a concept of highly intelligent agents that can be used for more complicated tasks. They can collaborate with each other through different types of interactions and attempt to form collaborative groups to increase capabilities, hence improving system performance. A coalition system can be a good example for such description. Cluster systems lie between the two ends of the spectrum. While such systems are more complicated structurally, they preserve the simple behaviour of each node. Although clustering makes the system difficult to manage, it increases flexibility, which is needed for restructuring of the system in a dynamically changing environment. For instance, if the required capabilities needed for an introduced task cannot be found within the current distribution of resources, new clustering may solve the problem.

Moreover, the similarities between the command and control hierarchy and Hierarchical Clustering structure make the cluster architecture more appropriate for marine security operations domain. However, in order to thoroughly address all the features of our system, different characteristics from the introduced models in this chapter have been exploited in our proposed system, which will be described in more detail in the following chapters.

Chapter 3

Problem Characteristics

In 2004, the Canadian government introduced a need to have marine operational centres to provide safety and security for the coastline of Canada. This motivation was published in the document entitled *Securing an Open Society: Canada's National Security Policy.* Therefore, the application domain of this thesis is mostly related to marine security operations. As defined in [2], there are considerable challenges for such centres.

> *A Marine Security Operation Centre (MSOC) provides support to operations by collecting vast amounts of information from the marine environment and analyzing it to create a maritime domain awareness picture. The primary purpose of an MSOC is to produce actionable intelligence, concentrating on national security, organized crime and other criminality and to communicate the information to the appropriate jurisdiction in a timely fashion.*

There are different government departments (such as National Defence, RCMP, Canadian Coast Guard, etc.) involved in an MSOC. The complexity of such centres can be best understood by considering the information exchange among the departments and the level of cooperation between the personnel, necessary for accomplishing a given task. Large volume of marine traffic and management of resources impose different challenges on such a system. Therefore, an intelligent decision support system [46] appears worthwhile for exploration as a choice of computational means to facilitate the complex command and control tasks performed by MSOC personnel by improving situational awareness and automating certain routine coordination tasks.

In a large volume of maritime traffic surveillance, there is a network of a huge number of distributed, heterogeneous and autonomous resources. These resources have different characteristics, they can be fixed or mobile, they can perform different tasks, they also have different constraints and capabilities. In addition, they are connected to each other by different means of communications. Let (V, E) be a graph of nodes. The nodes might represent different entities (e.g., resources, MSOC units, even sensors); the edges show connectivity (e.g., link, TCP/IP, radio) or relationships (e.g., Command, Control). A node is, therefore, an information producer, gatherer, processor, and/or consumer. The network edges represent the information exchange pipes and information fusion structures. Inherently, matching with the concept of command and control, such a network consists of a hierarchy of mobile resources operating under a given commander who introduces new missions into the system. Missions represent complex tasks, typically involving a number of explicitly or implicitly identified subtasks, each with specific resource capability requirements that need to be matched with the capabilities of mobile resources in order to perform a given task. In general, operations are carried out by allocating a set of resources to a mission. These resources are distributed in time and space. Therefore, coordination of these resources can be a challenging problem. In such a hierarchical network, each resource is assigned to different tasks over time by higher level commanders. Consequently, each commander gets his order from a higher authority. In our network we assume that MSOC, as the top-level node, is the highest level authority, with the ability to introduce new missions into the system.

Intuitively, the first step to propose the solution for a problem is to identify the problem characteristics. In the domain of maritime surveillance, there are three main problem characteristics that can be recognized and any proposed model should address these features: Distributed System, Dynamic Environment, and Uncertainty.

3.1 System Distribution

The system can be described as distributed in which there are a large number of heterogeneous resources working in different locations, asynchronously. Generally, the system consists of different units and sensors. The resources can be located on different platforms. For instance, consider a sensor installed on a helicopter, where the helicopter itself is located on a frigate and can fly a certain distance from frigate. Therefore, the complexity of the resource distribution can be easily recognized when an integrated resource management model is needed for such a system.

When a node receives a complex task, it tries to find a child node with matching capabilities to perform the task. If none of the child nodes can perform the task, it attempts to split the task into a collection of subtasks that can be performed by two or more of its child nodes. Of course, if this attempt fails as well, the task will be rejected. Otherwise, it will travel downward the hierarchy until all of the resulting tasks are of elementary type, meaning that each of them can directly be assigned to a single resource capable of performing the task.

Accordingly, the status of the original mission can be achieved upon completion or failures of the subtasks and should be reported back to the higher level nodes. Such flow of tasks in the system from the top level node (MSOC) downwards to the lowest level nodes (actual resources) is also called *Task Life Cycle* and will be discussed in more detail in the following chapters.

3.2 Dynamic Environment

We are interested in providing intelligent decision support for real-world operations. Any such support system exists in a dynamic environment, where changes occur outside of the system's control, e.g. weather. These changes can be categorized as one of two kinds: internal or external. Internal changes refer to the changes that happen inside the system. For instance, the set of tasks belonging to a particular mission may grow or shrink from one system state to the next, due to the dynamic nature of missions. Likewise, the total number of missions to be performed concurrently, in any given state, can be varied, hence making it difficult to predict and can cause sudden spikes. Thus, the overall workload and the global distribution of work within the system may change spontaneously and vary considerably over time. External changes, on the other hand, are caused from outside of the system. Resources can be problematic, as external conditions in the operational environment adversely affect the performance of a distributed fusion system. Reduced resource capabilities occur due to transient or permanent unit failures, as well as, common events in the physical environment in which the resources operate.

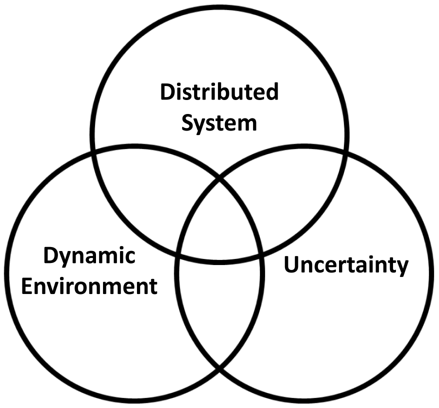


Figure 3.1: Problem Characteristics

3.3 Uncertainty

Uncertainty is another concern of a system that deals with real-world operations. There is an unavoidable lack of perfect knowledge in the system. The current state can never be known definitively, since all sensors have limitations. It is also impossible to be certain of the outcome of actions in advance, since there may be randomness or other factors involved. Therefore, fault tolerant behaviour is crucial for increasing the reliability of the system and to avoid catastrophic system failures that are created by communication failures and partial or total resource failures (e.g., in disaster situations). Since the knowledge in the system can be gained gradually, the decision making procedures should be able to adapt to new situations by providing partial decisions based on the incomplete knowledge.

The system we are dealing with, lies within the overlaps of the three main problems mentioned above (see Figure 3.1). In addition, operating under dynamically changing and essentially unpredictable conditions calls for design principles that ensure robustness and scalability. Therefore, such a surveillance system has two specific design challenges:

- Create a feasible set of combinations for an initial task allocation: the feasibility of a set of resource combinations has to be reviewed a priori to deal with high priority task execution or with routine surveillance operations.

- Adjust to changes: there are two dynamic control mechanisms to adjust to internal changes in resource requirements and external changes affecting the availability of resources.

a. New events may require a reconfiguration of resources in order to deal with an undesired network status, accommodate distributed fusion dynamic requirements, or address additional tasks triggered by new information in the network. The resulting configurations or structures have to be constantly refined.

b. It is also vital that any system has the capacity for re-planning and re-tasking. Re-tasking involves re-assigning a task to a different resource or set of resources, because the original assignment was unsuccessful. Re-planning is invoked when a plan, in its current formulation, is no longer able to provide a solution for the mission it was designed for. Changing situations or the unforeseen consequences of actions can result in requiring this kind of flexibility. An acute example of this is when rescue workers themselves, end up needing to be rescued.

Given the scale of systems considered here, specifically the enormous volume of data and information to be processed, the complexity of control tasks to be managed, and the dynamic nature of mission requirements [72, 75], a decentralized approach facilitates dynamic configuration and management of multiple mobile resources in an unstable environment and enhances the robustness and scalability of the distributed information fusion system, by avoiding the bottleneck of centralized fusion systems [63].

Chapter 4

Proposed Model

This chapter introduces a conceptual model shaping the NADIF system along with the description of its components. Following that, a reference model is introduced to describe a generic abstract scenario. The model also shows the correspondence to the three building engines previously mentioned (e.g. Resource Management, Decision Support, and Information Fusion). The choices made for this system reflect the challenges and limitations faced within the realm of marine safety and security. The proposed model uses some of the architectural design principles previously presented in [33].

4.1 High Level View

As illustrated in the previous chapter, circumstances can change rapidly. Therefore, operating under uncertain mission requirements in an unstable physical environment calls for flexible adaptation to new situations. In this regard, using collaborative self-organizing system concepts [42] naturally facilitates reconfigurable applications and dynamic reorganization in response to internal changes in resource requirements and external changes affecting the availability of resources. In addition, considering the complexity of introduced tasks and the volume of data and information in the system, a decentralized technique for management of resources and tasks, arguably enhances robustness and scalability of the system.

Therefore, the proposed model is built on a decentralized organization of mobile resources and the tasks they perform, one that naturally supports a hierarchical command and control structure. The complex command and control structure is realized in a distributed and hierarchical fashion by means of a dynamic ensemble of autonomously operating *control* *agents*, interacting with one another and with their local environment. Intuitively, each agent is associated with an individual resource representing a concurrent control thread of the decentralized system. Agents are created or eliminated at run-time as resources are added to or removed from the system.

In order to simplify mapping the tasks onto resources that execute them, the network architecture, hierarchically organizes resources into clusters. Control agents are nodes in the network architecture; their organization into clusters is stated by undirected edges (see Figure 4.1). Referring to distinct roles of resource entities, there are two different types of nodes that can be distinguished as follows:

- *Physical resources* refer to real-world resource entities as part of an existing distributed fusion system. In the hierarchical structure, only the leave nodes represent physical resources. Depending on the level of abstraction at which a distributed fusion system is considered, a physical resource may refer to a group of mobile sensor platforms, to a single mobile platform, or even to an individual sensor unit on a given sensor platform.

- *Logical resources* refer to abstract resource entities formed by clustering two or more physical and/or logical resources, each with a certain range of capabilities, into a higher level resource with aggregated (richer) capabilities needed to perform more complex operations. A logical resource identifies a *cluster* of resources.[1] In the hierarchical structure (e.g. as illustrated in Fig. 4.1) all non-leave nodes represent logical resources.

For increased robustness and to reduce control and communication overhead, logical resources operate semi-autonomously, making local decisions on the realignment and reorganization of resources within a cluster. Resource management policies govern the migration of resources between clusters based on common prioritization schemes for resource selection, load balancing, and organization of idle resource pools [33]. Reconfiguration is performed in an ad hoc manner using 'plug and play' mechanisms. Resources may be added or removed from a cluster on demand and depending on their sensor capabilities, mobility capabilities, geographic location, cost aspects and other characteristics. The underlying design principles resemble those for improving performance and robustness in mobile ad hoc networks [56].

[1]Logical resources require *command* capabilities in their clusters; we abstract away from this notion in this report.

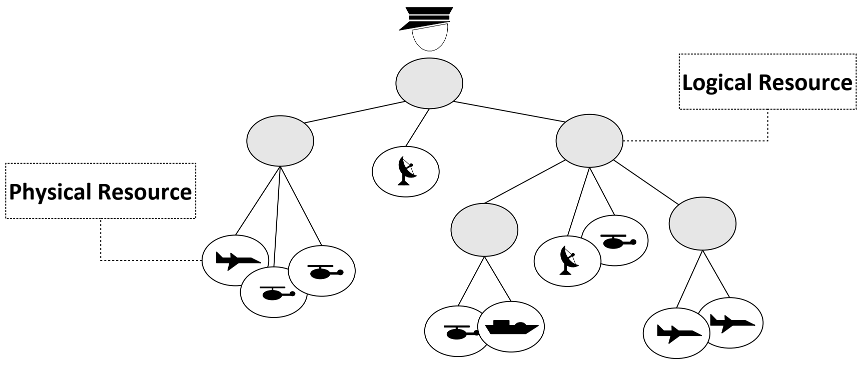


Figure 4.1: Nodes represent physical or logical resources.

Organization Principles

Specific challenges arise from complex interaction patterns between logical and physical resources and the dependencies between the operations and tasks to be performed in a collaborative fashion. The following organization principles outline some of the aspects that need to be addressed.

- *Resource Clustering Principles* control the arrangement of resources into resource clusters. Composition rules defined over resource descriptors specify the clustering of resources so as to form composite resources with richer behaviours. A resource descriptor is an abstract representation of resource attributes such as physical capabilities (e.g., sensor capabilities, mobility and time constraints), geographic position and workload information.

- *Resource Distribution Principles* refer to distribution of resources which can be either physical or logical:

  - *Physical distribution* is the spatio-temporal distribution of mobile resources in the geographical environment. Position information and projections of resource trajectories provide important input for grouping resources into clusters (e.g., keeping resources of the same group in close proximity to each other) and also to satisfy communication requirements (e.g., moving a resource in order to act as a communication proxy).

- *Logical distribution* refers to the dynamic configuration of physical resources into clusters that change in response to the changes in tasking information (e.g., new task orders or changes in task priorities), changes in the capabilities of resources (e.g., device failures or new resources joining the network), changes in the environment (e.g., changes in weather conditions), and to maintain a desirable workload balance within individual clusters and across the whole network.

Resource Allocation

Missions represent complex tasks, typically involving a number of explicitly or implicitly identified subtasks, each with specific resource capability requirements that need to be matched with the capabilities of mobile resources in order to be performed. In general, the operations carried out by a set of resources, allocated to a mission, are distributed in time and space, making the coordination of these resources a challenging problem.

In the domain of marine safety and security, new missions are introduced by assigning a mission to the top-level node. When a logical resource receives a complex task, it tries to find a child node with matching capabilities to perform the task. If none of the child nodes can perform the task, the logical resource attempts to split it into a collection of subtasks that can be performed by two or more of its child nodes. Intuitively, one may view new tasks as 'sinking' downwards in the node hierarchy until they reach a matching physical resource or become transformed into a collection of related subtasks.

Since our system takes place in a dynamic environment where actions have uncertain outcomes, it must be able to replan an alternative solution to a mission when the need arises. In order to be able to know when to do this, the system should evaluate the outcome of finished tasks.

4.2 System Architecture

As mentioned before, there are three engine blocks introduced in our system. Resource Configuration Management is responsible for monitoring and configuration of resources by dynamically updating the command and control hierarchy, so that for every task introduced to the system, it provides a list of capable resources which can be used to perform the task. This engine is based on the work done by Farahbod et al. [33]. In their research, a Dynamic Resource Configuration Management Architecture (DRCMA) is designed so that the best matching resources with minimum cost can be found by use of a bidding algorithm to quote from different child nodes [29]. Therefore, the Resource Management engine of the proposed model in this thesis is an extension of their work.

Another important building block of the system presented here is Decision Support; in order to support decision making of the system operators, it is vital for the system to be able to simulate the entire process of decision making. In our system, the main responsibility of the Decision Support engine is organization and management of tasks. Therefore, we define Task Management as the the process of handling and monitoring the missions from the moment they are introduced to the system until their completion or failure [31].

Finally, the basic aspect of the Information Fusion engine is implemented by gathering information and updating the current state of the system. Other fusion aspects (e.g. transforming raw data into knowledge) are not relevant to this thesis.

In addition to the three engine blocks mentioned above, the system can also be decomposed into four components in the design level [43]. These components are based on the four aspects of mission completion that our system is required to process: planning, resource management, execution management and tasking. Isolating these aspects into individual areas of responsibility has facilitated greater separation of concerns. This has several benefits: complexity of the system as a whole is reduced; design and construction of the system are expedited; design validation becomes more straightforward; and primary interactions become apparent and can be fully addressed. Each of the four aspects corresponds to a component in the model of our system: the Planner, Resourcer, Executor and Tasker, as illustrated in Figure 4.2.

It is worth mentioning that each component at design level is related to an engine in the system. The Resourcer is responsible for the role of Resource Configuration Management engine, while the Decision Support responsibility, management of tasks, can be fulfilled by Planner, Tasker and Executer. The information fusion is achieved by situation assessment on the updates received from different components.

In addition, the behaviour of each component is represented as an *intelligent agent* in the system, operating in a coordinated fashion [46]. In this section general responsibilities of these components are described, leaving the functionalities and their interactions to the next section.

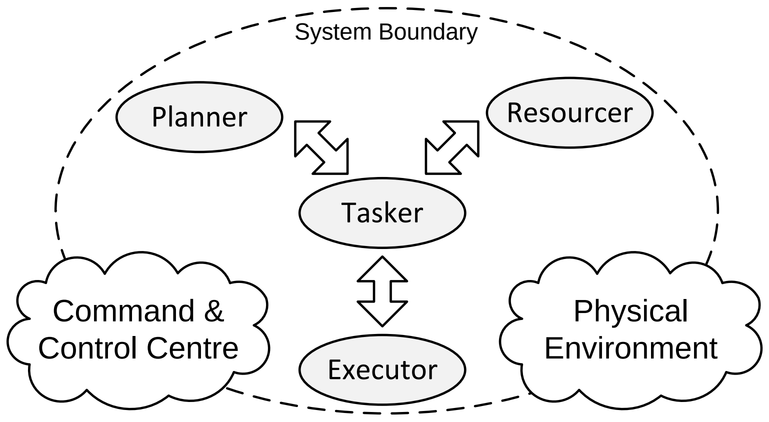


Figure 4.2: Components of the System Architecture

4.2.1 Planning

The main goal of the system is to successfully complete the missions introduced to it. A mission, as stated, is usually too abstract to be dealt with, so it must be broken down into a set of executable actions. Automated planning is the field of computing that deals with this process [66]. The process of planning involves finding a set of actions that will accomplish a set of goals, without violating any relevant constraints. For example, precedence constraints are used to establish which tasks precede others. Since our system resides in a dynamic environment where actions have uncertain outcomes, it must be able to replan an alternative solution to a mission when the need arises. The Planner must be able to evaluate the outcome of a finished task so that it will know when to do this.

We employ the Hierarchical Task Network (HTN) approach to generate plans [26]. HTNs work by successively refining abstract tasks to increasingly concrete tasks using replacement rules called methods. The tasks are maintained in a tree-like structure that depicts both the relation between abstract parent tasks and refined subtasks as well as constraints between the tasks. If a solution to the mission can be found, all of the leaves of the graph will be executable tasks, which are called primitive tasks [38]. One advantage of using HTNs is that the methods mirror domain expert understanding. Thus plans generated by an HTN system make sense in their relevant field, and verifying the logic of an HTN planner is straightforward for someone familiar with that field. There are several popular implementations of the HTN approach available, such as JSHOP2 [50]. Furthermore, HTN plans are amenable to re-planning, including local re-planning, which changes as little as possible in a problematic

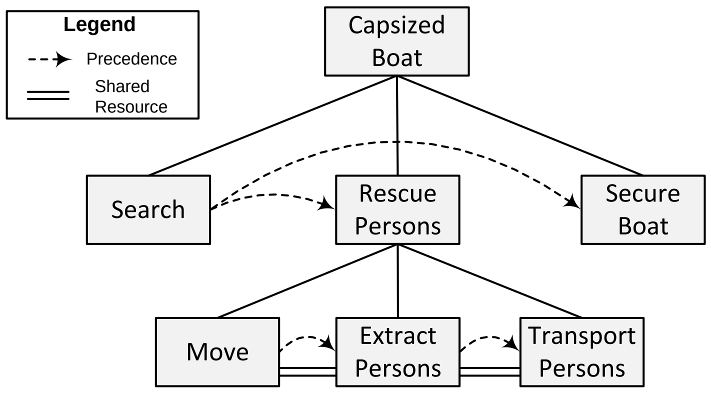


Figure 4.3: Hierarchical Task Network (HTN) Example

plan [8].

A simple example of how an HTN planner works can be shown with the following example (see Figure 4.3). The mission, *Capsized Boat*, is introduced to the system as an abstract task. This task is too general to be processed so we start to decompose it using the *Search and Rescue* method, which breaks it down into three tasks: *Search*, *Rescue Persons* and *Secure Boat*. Notice that precedence constraints exist between *Search* and the other two tasks: *Search* must be completed first, then the others may proceed in any order. In the diagram, *Rescue Persons* has been refined further into more specific subtasks: *Move*, *Extract Persons* and *Transport Persons*. This process of refinement through the use of methods can continue with any leaf node in the network until all the tasks generated are primitive.

4.2.2 Resource Management

For any conceivable real-world task, one or more resources are required to achieve it. For example, a marine patrol aircraft may be necessary to perform an air search, a helicopter to extract a person from sea, and a paramedic to administer first aid. Our decision support system must keep tabs on which resources are available in the system, and be able of establishing which of those would be appropriate for dealing with any given task. It is important to note that there may be multiple criteria for determining which resource is most appropriate for a specific job, such as time and cost. Thus the system should be able to come up with a selection of resources that are capable of fulfilling a task, even though they may do it in different ways and with different levels of efficacy.

As mentioned before, the Resourcer component, is responsible for configuration and management of resources within the system. The assignment of resources to given tasks can be improved by orchestrating them into different clusters. This organization in the system is also aligned with the C2 hierarchy, and the hierarchical clustering structure previously mentioned. In other words, the resources should be arranged into groups that are complementary in terms of characteristics and capabilities. This arrangement should be done dynamically, taking into account the current conditions of both the resources and the external environment (e.g. weather).

4.2.3 Execution Management

Execution Management concerns the handling of tasks that are currently underway. While the actual undertaking of the task will be done by the resources assigned to it, the decision support system is also required to perform certain management activities. The progress of task execution must be monitored. In addition, interactions between a task and the rest of the system must also be handled. For example, a report will be generated and passed along when a task finishes. Similarly, messages may be sent to or from the task in problematic circumstances.

4.2.4 Tasking

During design, we recognized that in addition to planning and execution management, there are other states that tasks can have. We use the concept of Tasking, common in defense-related literature, to deal with this issue. Tasking includes responsibilities such as ensuring that resources are properly assigned with no conflicts, and execution constraints, which determine when a task executes, are observed. The Tasker is also employed to manage interactions between the other components. In this way, we can deal with the timing issues that can occur in a real-time asynchronous system.

The Tasker is also responsible for deciding what happens to a task when finishes executing. Depending on the generated execution report, the task may be retried with or without a new resource assignment. A task may also be sent to the Planner for evaluation, which could result in re-planning, or the generation of a mission report.

4.2.5 Decentralized Control

This system is intended to be implemented in a distributed manner, with all four services running on each node in the C2 hierarchy. This allows for a truly decentralized control structure and improves the robustness of the network. However, the distributed implementation of the system will be described in more detail in the next chapter, but for now, the examples are given using a centralized model, where a single set of component services organizes all resources in the network.

4.3 Reference Model

In the previous section, the four main areas of responsibility were described. We now describe the interactions between these components in terms of an abstract generic scenario indicating their responsibilities and relations.[2] In order to model and analyze the scenarios of the system, a standard notation, *User Requirements Notation (URN)* [61, 5] is used.

4.3.1 An Introduction to User Requirements Notation

In 2008, the *User Requirements Notation* was approved as an *ITU-T* standard [64] that combines concepts and notations for modeling goals and scenarios. *jUCMNav* [61] is an Eclipse plug-in for *URN* used here to model and analyze scenarios.

A *scenario* describes a partial usage of the system. It is defined as a set of partially ordered responsibilities to be performed such that the system reaches its goals. Each scenario has *start points*, represented by filled circles, which capture the preconditions or triggers of the system. *End points* are illustrated by solid bars and capture the postconditions of the system. A scenario progresses along *paths* between these start and end points, and the *responsibilities* are represented by crosses on the path. The diamond symbol is called a *stub* and is a placeholder for a sub-scenario. We employ them in our model for complexity management which can be done by encapsulating some related and coherent responsibilities in a *subcomponent*. Beyond concepts and notations mentioned here, there are some other aspects supported by *URN* but they are not used here.

[2]For the sake of simplicity, the scenario is given in a centralized fashion.

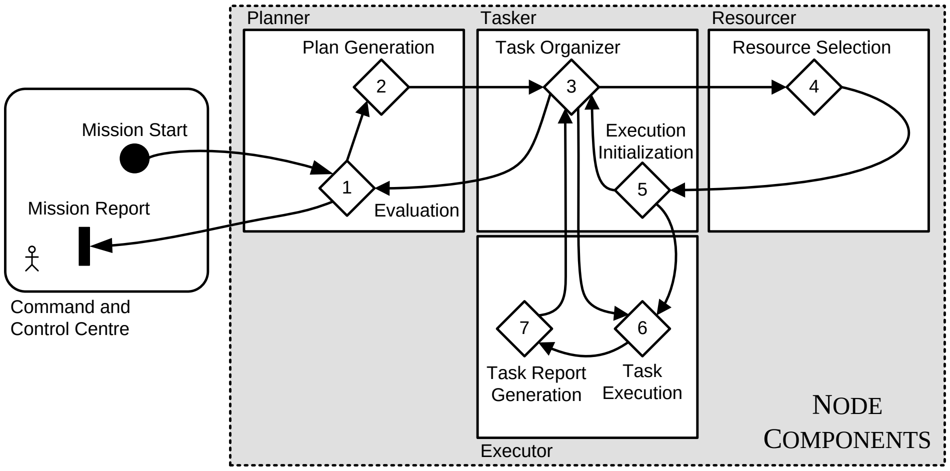


Figure 4.4: Abstract Generic Scenario of the NADIF System

4.3.2 Describing the Abstract Generic Scenario

We use *jUCMNav* for modeling different concrete scenarios of the system in various cases and situations. The *abstract generic scenario* is the result of generalizing the common parts of these concrete scenarios. As shown in Figure 4.4, the system has four components in addition to the command and control centre. This section defines the responsibilities of each component and also the communication among them. It is important to note that Figure 4.4 shows only the flow of control and the duties of each element, therefore, some parts of the path can be executed concurrently for different missions or tasks.

The command and control centre is outside the boundary of the system and is assumed to be an actor on the system. This component is responsible for introducing a *new mission* to the Planner. In addition, it will receive a report of the finished mission, whether it is successful or not.

1. *Evaluation*: This subcomponent is responsible for evaluating finished tasks. If the results of a task compromise an active mission, it is sent back to Plan Generation for re-planning. If instead, it means that a mission has finished (whether successful or not), this subcomponent issues a report to the command and control centre.

2. *Plan Generation*: In this subcomponent, the current plan is decomposed into a set of tasks. Any tasks that are *atomic*, and thus executable, are sent to the Task Organizer subcomponent in order to wait for resource assignment and execution.

3. *Task Organizer*: This subcomponent maintains the pool of ready tasks that are waiting for execution. When possible, it chooses one of these tasks and checks its status. If the task needs a resource, a request is made. If the task is ready to execute, then it is sent to Task Execution. This subcomponent also checks if a task can no longer be executed, due to exceeding its time window, finished status (from the execution report sent by Task Report Generation), or if no resource assignments are possible (i.e., a rejection message from Execution Initialization). In these cases, the task will be sent back to Evaluation, which may result in re-planning if necessary.

4. *Resource Selection*: This subcomponent acts as a filter to find the resources that satisfy the required capabilities of a task. In this manner, a list of resources that are able to perform the task is created and sent to the Execution Initialization subcomponent in the Tasker.

5. *Execution Initialization*: Its main duty is to pick the best resources for executing the current task from the list provided by Resource Selection. The decision is based on different parameters such as resource availability, task priority, resource location, and other information. First, resources currently in use by tasks of equal or higher priority are pruned from the list sent by Resource Selection. If the resulting list is empty, there are no appropriate resources for executing the task, and an exception is sent to Task Organizer. Otherwise, one or more resources from the list are assigned to the current task, which is in turn sent to the Task Execution subcomponent for execution. If a selected resource is in use, a request to release the resource is sent to the Task Execution subcomponent.

6. *Task Execution*: This subcomponent is responsible for monitoring the execution of the current tasks. In addition, it will release any resources that are required by higher priority tasks on request.

7. *Task Report Generation*: Whenever an task execution has finished, this subcomponent generates a report. The report contains the results of task performance and it should be sent to the Task Organizer subcomponent. This report will be used in Task Organizer to determine whether or not the task has effectively finished execution, and in Evaluation to determine the effect of any execution on the progress of the mission as a whole.

Chapter 5

Formal Model

This chapter introduces a precise architectural model defined in terms of a Distributed Abstract State Machine (DASM) based on the asynchronous ASM modeling framework. Building upon the design concepts described in Chapter 4, the formal description of the model can turn the abstract architecture into a high-level computational model that can be systematically analyzed, inspected, refined and validated. The formal representation ensures that the key system attributes are specified concisely and unambiguously, providing a reliable foundation for checking that these attributes are well understood and actually do meet the functional requirements.

In this chapter, we present another parallel view of the system. In this perspective, the model consists of Resource Management and Task Management. The former is responsible for dynamic configuration of the resources by using hierarchical clustering structures while the later is responsible for the management of the assigned tasks. It should be mentioned that the focus of this thesis is mostly related to the Task Management. The Resource Management part is based on the Dynamic Resource Configuration Management Architecture presented in [30, 33].

In this regard, by using the CoreASM modeling environment [28], the specifications of the proposed model have been described in DASM.[1] We also developed a basic graphical user interface in Java to provide a live view of the resource network and its command and control hierarchy during the simulation of the scenarios using the JASMine plugin of CoreASM [34].

[1]The current version of our model in CoreASM is available for download in the Software Technology Lab website at `http://www.stl.sfu.ca`.

The structure of this chapter is as follows. A formal description of the system's basic concepts is given in the first section. Next, the Task Management is presented in more detail. Finally, a simple scenario is presented to show how the system works.

5.1 Basic Concepts

In order to define any system precisely, the elements should be formally described. As mentioned in the previous chapters, our system consists of a hierarchical network of resources, which is aligned with the command and control hierarchy used in military. Such a structure was depicted in Figure 4.1. In the following, some of the main elements of the proposed model is described with use of Abstract State Machine formalism.

5.1.1 Node

One of the basic elements in our model is *Node*. Each node refers to either a logical or a physical resource in the network (see Section 4.1), where logical resources represent clusters of resources formed to perform coordinated tasks (e.g., in a search and rescue mission). Every logical resource represents a non-empty set of subordinate resources, called *child resources*, that belong to its cluster. We formally model this structure as follows:

Resources

universe $\textsc{Node}$
universe $\textsc{Resource}$

$\mathit{node} : \textsc{Resource} \rightarrow \textsc{Node}$
$\mathit{resource} : \textsc{Node} \rightarrow \textsc{Resource}$
$\forall r \in \textsc{Resource}\ \forall n \in \textsc{Node}\ \ \mathit{resource}(n) = r \Leftrightarrow \mathit{node}(r) = n$

$\mathit{cluster} : \textsc{Resource} \rightarrow \textsc{Resource-set}$
$\mathit{isCluster} : \textsc{Resource} \rightarrow \textsc{Boolean}$
$\forall r \in \textsc{Resource}\ \neg \mathit{isCluster}(r) \Rightarrow (\mathit{cluster}(r) = \{\})$

5.1.2 Capability

Every resource identifies a nonempty set of *capabilities*. For each capability, there is a set of properties, defined as pairs of ⟨*name, value*⟩, that specifies the particular types of services that the resource provides (name) along with related quantifying measures (value).

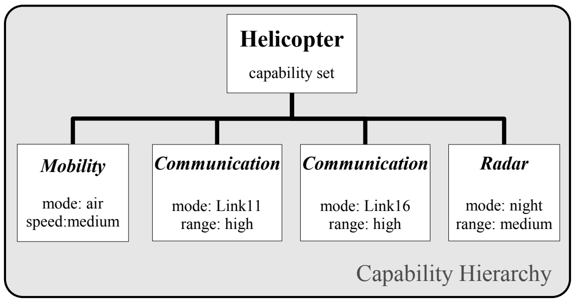


Figure 5.1: Capability Hierarchy example

As illustrated in the Figure 5.1, a helicopter has 'mobility', 'communication', and 'radar' capabilities. The 'mobility' has some properties such as: $\langle$*speed, 'medium'*$\rangle$, $\langle$*mode, 'air'*$\rangle$.

Capabilities

universe $\textsc{Capability}$

$capName : \textsc{Capability} \rightarrow \textsc{String}$
$capProperties : \textsc{Capability} \rightarrow \textsc{String} \times \textsc{Value}$

Resource capabilities refer to operational aspects such as sensor capabilities, mobility constraints, time constraints, et cetera; as such, they are subject to dynamic changes caused by impediments like weather conditions and unit failures that may restrict temporarily or permanently the full range of services that a resource can potentially provide.

In addition, resources can have more than one instance of a capability. Imagine that the helicopter in our example can communicate with two different communication links. Therefore, it has two 'communication' capabilities, each with a different value for its 'mode' property, such as $\langle$*mode, link11*$\rangle$, and $\langle$*mode, link16*$\rangle$ (see Figure 5.1).

The capabilities of a cluster are the aggregated capabilities of its resources in combination with the additional capability of orchestrating these resources as required to perform certain tasks or missions. Thus, higher level logical resources usually have more complex or sophisticated capabilities (e.g., *search and rescue*), which are composed of lower level capabilities of the resources that belong to their clusters. See also Figure 4.1 for an example.

Capabilities

$resourceCapabilities : \textsc{Resource} \rightarrow \textsc{Multiset}(\textsc{Capability})$
$capabilities : \textsc{Resource} \rightarrow \textsc{Multiset}(\textsc{Capability})$

$\forall r \in \textsc{Resource} \;\; capabilities(r) = resourceCapabilities(r) \cup (\bigcup_{s \in cluster(r)} capabilities(s))$

The relationship between a physical resource and its capabilities is formally described by means of a unary monitored function *resourceCapabilities* that, in any given system state, associates some finite set of capabilities with each of the elements in Resource. Likewise, a function *capabilities* is defined on each resource. This function identifies a set of aggregated capabilities for each of the logical resources, based on the respective resource cluster they belong to.

Logical Distribution The logical distribution of resources is reflected by the network topology as stated through the following functions defined on nodes:

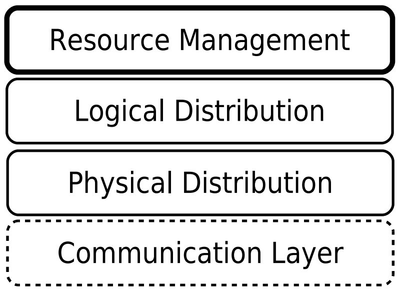
- $rootNode : \;\; \rightarrow \textsc{Node}$, a static function that identifies a distinguished node of the network representing the top level command and control unit.
- $childNodes : \textsc{Node} \rightarrow \textsc{Node-set}$, $childNodes(node)$ holds the set of nodes under direct authority of *node*.
- $parentNode : \textsc{Node} \rightarrow \textsc{Node}$, points to the parent node of a node.

Physical Distribution The physical distribution of resources within a given geographical environment in which they operate typically changes over time. This is abstractly modeled by a monitored function

$$location : \textsc{Resource} \rightarrow \textsc{Coordinate}$$

which, in any given system state, associates with each of the resources a geographical location, e.g. as identified by a global positioning system. As resources change their location dynamically, the function *location* may change its interpretation from state to state. Based on the location of resources and certain dynamic characteristics of the environment, such as weather and terrain conditions, various derived functions model various aspects of the physical distribution of resources. For instance, for any given node, the set of all the given nodes that are reachable over a certain communication channel is modeled by

Figure 5.2: RMA is organized in four layers.



$$\mathit{visibleResources} : \textsc{Resource} \times \textsc{Channel} \rightarrow \textsc{Resource-set}.$$

5.1.3 Layered Architecture

In the Dynamic Resource Configuration Management Architecture (DRCMA) presented in the previous works [33, 30], resource management policies govern the migration of resources between clusters based on common prioritization schemes for resource selection, load balancing and organization of idle resource pools. Configuration and management of resources across nodes is organized by means of a service-oriented architecture consisting of four hierarchical service layers, namely: Resource Management (L_4), Logical Distribution (L_3), Physical Distribution (L_2), and Communication (L_1), where L_1 refers to the bottom level layer; see Figure 5.2. The proposed DRCMA model assumes clearly identified and well defined interfaces between layers, such that $layer_n$ renders services to the next higher $layer_{n+1}$ using $layer_n$ protocols realized by means of services provided by $layer_{n-1}$ [40]. The encapsulation of services in separate layers not only improves the separation of concerns but also simplifies the control of complexity by providing convenient abstractions for decomposing complex interaction patterns.

The physical distribution layer provides services to query about and manipulate the physical distribution of resources in DRCMA. We abstract here from the internals of this layer, assuming an underlying model resembling those used in the routing layer of mobile ad hoc communication networks [40].

5.1.4 Capability Pattern

Most often a task needs more than one type of capability to be performed. To address this need, we define *Capability Patterns* with almost the same structure of the capability, but with the conceptual difference that the property values of capability patterns can be a range or a set of acceptable values. For instance, suppose an introduced task requires ‘search’ from either ‘air’ or ’ground’. Therefore, any resource which has one of these two property values in their capabilities is a candidate to perform this task. Here, capability pattern can be used to find the capable matches among the resources.

A *matchCapability* function of the form

$$\mathit{matchCapability} : \textsc{CapabilityPattern} \times \textsc{Capability} \rightarrow \textsc{Boolean}$$

holds if a capability matches a given capability pattern.

Capability Pattern

domain $\textsc{CapabilityPattern}$

$\mathit{requiredCaps} : \textsc{Task} \rightarrow \textsc{Set}(\textsc{CapabilityPattern})$
$\mathit{decomposeCapability} : \textsc{CapabilityPattern} \rightarrow \textsc{Set}(\textsc{CapabilityPattern})$

requiredCaps is a set of all the required capabilities in the form of capability patterns. These are used for matching the capabilities needed for the given task. Capability patterns can also be viewed as higher level capabilities used in logical nodes in order to find matching lower level resources for the introduced tasks.

Platforms

A group of physical resources may be physically bound together on a *platform*. A boat or a helicopter with various sensors are two examples of platforms. Resources on a platform, together with the platform itself, are moved together and their location is relative to platform’s absolute location. Each platform has a set of resources and each resource has at most one platform. The following function maps a resource to its platform.

$$\mathit{platform} : \textsc{Resource} \rightarrow \textsc{Platform}$$

If resource r does not have a platform, we have $\mathit{platform}(r) = \mathit{undef}$. If we count platforms as resources as well, we can have a hierarchy of platforms. A resource sitting on a platform could itself be a platform for other resources. For example, a helicopter is a platform for its various sensors, while it can be, as a whole, a resource which is landed on a frigate.

It is important to note that the concept of platform is different from that of a logical resource. A logical resource represents a cluster of resources that are, in most cases, logically suitable to perform a specific task. Logical resources also aggregate resources based on the command and control hierarchy, but not based on the physical binding of resources. To separate these concepts, the resource-platform relation is not automatically mapped onto the logical distribution of resources. As a result, we can potentially have a resource in the system (e.g. radar on a frigate) that is controlled by a commander unit different from the resource's platform.

There are some characteristics that the concept of platform brings into our architecture:

1. Set of resources located on a platform can only be moved together. However, these resources that have the capability of 'mobility', but cannot move separately (e.g. a special airplane that should be controlled from a frigate) can still have a relative movement around their platforms. The actual location of such a resource can be computed based on their platform's location. We define $\mathit{coord}(r)$ to be the absolute location of resource r, and $\mathit{position}(r)$ to be the relative location of r to its platform. If a resource r' does not have a platform, $\mathit{position}(r')$ holds the absolute location of r'; one can read that as a position relative to $(0, 0)$.

$$\forall r \in \textsc{Resource} \quad \mathit{coord}(r) = \begin{cases} \mathit{coord}(p) + \mathit{position}(r) & \text{if } \mathit{platform}(r) = p; \\ \mathit{position}(r) & \text{otherwise.} \end{cases}$$

2. We define a *movement range* for resources that can move relative to their platform, as follows:

$$\mathit{movementRange} : \textsc{Resource} \rightarrow \textsc{Radius}$$

   In order to find out which resource is capable of reaching a target, we can compute resource movement range and add it to its platform location to see if it can reach target's location. If resource r is attached to a platform p (i.e. $\mathit{platform}(r) \neq \mathit{undef}$) and r doesn't have the 'mobility' capability, then its movement range is equal to the range of its platform. If r has the 'mobility' capability, the combination of $\mathit{movementRange}(r)$ and $\mathit{movementRange}(p)$ tells us how far r can go.

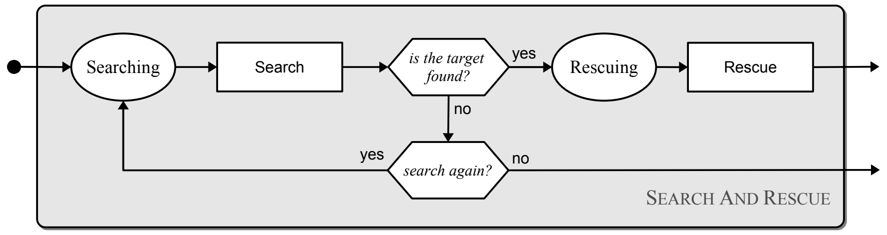


Figure 5.3: Control state diagram of a Search-and-Rescue task

Detachable Resources: We assign a *detachable* capability to model the possibility of a resource to be detached from its platform. A resource with a 'detachable' capability has the potential to move separately from its platform; i.e., it can have its own location and there would be no physical constraint on its movement since it is detached from its platform. In our example a helicopter on a frigate, the helicopter has the 'detachable' capability which allows it to fly away from the frigate.

5.2 Task Management

In our proposed model, high-level (or abstract) tasks are assigned to higher level nodes in the command and control hierarchy (see Section 4.1). As long as there exists a single child node or a combination of children capable of performing the task, the whole abstract task should bubble down the tree. In a situation where there is no combination of children that can perform an abstract task, the task is decomposed into subtasks.

The result of task decomposition should not only provide a *set of subtasks* the abstract tasks is comprised of, but also the *process* of orchestration of the subtasks.

Another issue which should be taken into account is that we may want to have some tasks unordered in a sequence (i.e. none of them has any priority to the others) so that, when the sequence is to be performed we may have to sort those tasks with respect to the availability of resources, time constraint, work balance and etc. In order to model this situation, *partially ordered* sets can be useful. In a partially ordered sequence, some of the elements might have been left unordered so that there is no priority between those elements. This can be used in *task decomposition* to build a sequence of tasks which may have some unordered elements.

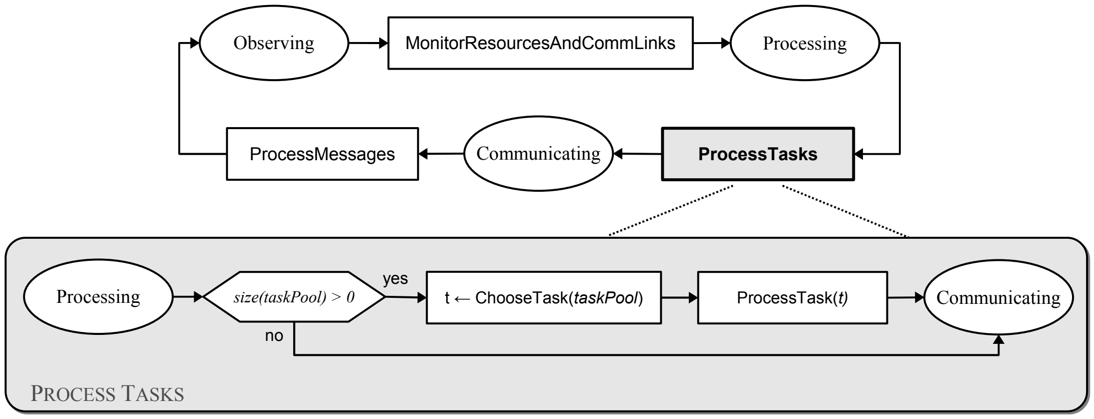


Figure 5.4: Activity cycle of a node (top) and control state diagram of ProcessTasks (bottom)

Such a decomposition can be modeled as a control state ASM diagram in which rules (boxes) represent subtasks. Every task starts with an input (normally assigned by the higher-level node) and generates an output that can be used as an input for the next task in the sequence. The output of performing the last task in a task decomposition of abstract task t is taken as the output of execution of t.

To give an example, an abstract *search-and-rescue* operation can be decomposed into two tasks of *search* and *rescue* orchestrated by the control state diagram of Figure 5.3.

5.2.1 Node Program

The process of performing tasks and dynamically maintaining the configuration of resources is modeled as a distributed process carried out by individual nodes of the network. Every node continuously goes through a cycle of three activity phases (see Figure 5.4): *observing*, where it monitors and observes its resources and communication links in order to update its "understanding" of the network and react to changes if needed, *processing*, where it processes and performs tasks that are assigned to it, and *communicating*, in which it processes and responds to messages received from other nodes in the network.

Here, we focus on the *processing* phase; we look into the ProcessTasks rule and explain how tasks are analyzed and eventually carried out by nodes.

5.2.2 Task Lifecycle

Every task, once injected into the network, goes through a lifecycle from being assigned to a node to being completed or rejected (see Figure 5.5). A task, once created, will always be assigned to a node—i.e., residing in the task pool of one and only one node at any time—and it will be in an implicit *waiting* state until it gets processed by the node it is assigned to. Tasks can be moved from parent nodes to child nodes and vice versa. The state (or "mode") of a task in its lifecycle is maintained by the nodes that are processing it through its lifecycle.

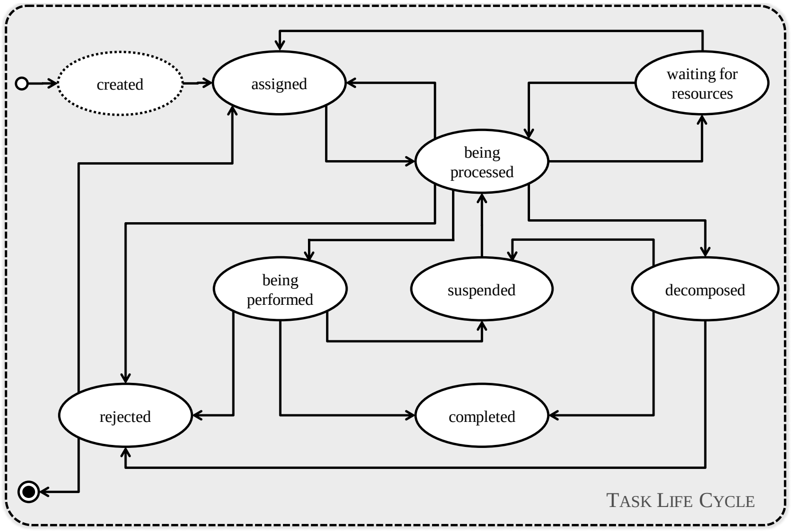


Figure 5.5: Task Lifecycle

Every node in the model, during its computation cycle, non-deterministically chooses a task t from $\mathit{taskPool}(\mathit{self})$, the pool of tasks that are assigned to it, and processes that task by calling the ASM rule $\mathsf{ProcessTask}(t)$;[2] (see Figure 5.4). Every call to $\mathsf{ProcessTask}(t)$ can potentially change the mode of task t, moving it to a different stage of its lifecycle (Figure 5.5). If the task is at the beginning of its cycle, ProcessTask will initialize the task and changes its mode to *beingProcessed*.

[2]Here, for the sake of simplicity, we assume that an abstract rule ChooseTask (see Figure 5.4 (bottom)), would non-deterministically choose a task. However, in further refinements it can take into account various factors such as the task priorities and their pre-conditions.

```
$NodeProgram$ **ProcessTask**$(t) \equiv$
  case $taskMode(t)$ of
    $assigned \rightarrow$
      InitTask$(t)$
      $taskMode(t) := beingProcessed$
      $\ldots$
```

When a task is in the *beingProcessed* mode, the processing node is basically looking for the suitable resources to assign to the task:[3]

```
    $beingProcessed \rightarrow$
        if $nodeSatisfiesCapabilities(self, t)$ then
          if $\neg leafNode(self)$ then
            $taskBiddingCtrlState(self, t) := requestingQuotes$
            FindMatchingChildNode$(t)$
          else
            if $allCapabilitiesAvailable(self, t)$ then
              AssignCapabilitiesToTask$(self, t)$
              $taskMode(t) := beingPerformed$
            else
              let $victimSet \leftarrow$ ComputeTasksToBeSuspended$(self, t)$ in
                if $|victimSet| > 0$ then
                  SuspendTasks$(victimSet, t)$
                else
                  $taskMode(t) := assigned$
        else
          $taskMode(t) := rejected$
          $taskRejectionReason(t) :=$ "Cannot match required capabilities."
```

If the node meets the capability requirements of the task:

1. If the node is not a leaf node, it starts looking for a suitable child node or a combination of child nodes that can carry out the task; this is done by setting the value of the control state function $taskBiddingCtrlState(self, t)$ to *requestingQuotes* so that from now on the task will be passed to FindMatchingChildNode rule until its task mode is changed. It should be mentioned that the focus of the Task Life Cycle is only on the 'mode' of a task; while Figure 5.6 illustrates the control state diagram of *FindMatchingChildNode* rule. This rule is responsible to find a proper child node capable of performing the

[3]In these rules, *self* refers to the node executing the rule.

task. The process of finding matching child node shown in Figure 5.6 involves sending quote requests to all the child nodes, waiting to receive the responses and decide whether to [33]:

(a) assign the task to a single child node, hence moving the task to the task pool of the child node and changing the task mode back to *assigned*;

(b) decompose the task into subtasks, hence changing the task mode to *decomposed* and create new subtasks that would start their lifecycle from being *assigned* to the node;

(c) aggregating required resources into a cluster that can later carry out the task, hence changing the task mode to *waitingForResources*;

(d) or rejecting the task if none of the above applies and changing the task mode to *rejected*.

2. If the node is a leaf node and if all the capabilities required to perform the task are available, they will be assigned to the task and the task mode becomes *beingPerformed*; otherwise, if some capabilities are busy with lower priority tasks, those tasks will be suspended to free up the required capabilities. In order to suspend any task, first a *victim* set should be defined. This set includes all busy tasks that have lower priority with regard to the priority of the given task. When such list is ready, then a task should be chosen from the list and all the resources previously assigned to it must be released for further assignments. If there is no task to be suspended, the task mode will be switched back to *assigned* and it stays in the task pool to be processed at a later time.

In the unlikely event that the node cannot afford the capabilities required by the task,[4] the task will be rejected. If the rejecting node is not a root node, the task will be sent back to the parent node.

When a task is decomposed, it is basically waiting for its subtasks to be completed. If at least one of the subtasks is rejected for any reason, the decomposed task will be considered rejected as well.[5] The following piece models this behaviour:

[4]This should not happen too often, since the parent node should have already matched the required capabilities of the task with the capabilities of the node; however, as we mentioned before, in a dynamic environment the capabilities of the node can change anytime.

[5]With further refinements, the node can try to re-allocate the rejected subtask to some other nodes.

```
$decomposed \rightarrow$
  **if** $\forall t' \in subTasks(t) \quad taskMode(t') = completed$ **then**
    $taskMode(t) := completed$
  **if** $\exists t' \in subTasks(t) \quad taskMode(t') = rejected$ **then**
    $taskMode(t) := rejected$
    $taskRejectionReason(t) :=$ "At least one sub task failed."
    CancelAllSubtasks($t$)
```

When a task is rejected or completed, it is removed from the task pool of the node, its resources are released and the parent node is notified of the rejection or completion of the task.

```
$completed \rightarrow$
  **remove** $t$ **from** $taskPool(self)$
  ReleaseCapabilities($self, t$)
  **if** $parentTask(t) \notin taskPool(self)$ **then**
    SendCompletionNotice($parent(self), t$)
```

Every node has to keep the following info:

$assignedChildNode : \textsc{Node} \times \textsc{Task} \rightarrow \textsc{Node}$

$taskAssigned : \textsc{Node} \times \textsc{Capability} \rightarrow \textsc{Task}$

Note: capabilities will be locked when they are being transferred. They will be unlocked as soon as they are assigned.

So far, some of the possible modes that a task can have during its lifecycle have been explained. In the following, the remaining task modes of the ProcessTask rule, presented in Figure 5.5, are described with ASM formalism.

```
$NodeProgram\,\textsf{ProcessTask}(t) \equiv$
  **case** $taskMode(t)$ **of**
    $assigned \rightarrow$
        $\ldots$
    $beginProcessed \rightarrow$
        $\ldots$
    $beingPerformed \rightarrow$
      **if** $taskCompleted(t)$ **then**
        $taskMode(t) := completed$
      **if** $taskFailed(t)$ **then**
        $taskMode(t) := rejected$
        $taskRejectionReason(t) :=$ “Task failed.”
    $suspended \rightarrow$
      **if** $suspensionCause(t) \notin taskPool(self)$ **then**
        $taskMode(t) := beingProcessed$
    $decomposed \rightarrow$
        $\ldots$
    $rejected \rightarrow$
      **remove** $t$ **from** $taskPool(self)$
      ReleaseCapabilities$(self, t)$
      **if** $parentTask(t) \notin taskPool(self)$ **then**
        SendRejectionNotice$(parent(self), t)$
    $completed \rightarrow$
      **remove** $t$ **from** $taskPool(self)$
      ReleaseCapabilities$(self, t)$
      **if** $parentTask(t) \notin taskPool(self)$ **then**
        SendCompletionNotice$(parent(self), t)$
    $waitingForResources \rightarrow$
      **if** $waitCause(self, t) = waitingForCapTransfer \wedge allCapabilitiesTransferred(self, t)$ **then**
        AssignTaskToNode$(t, newClusterNode(self, t))$
where
  $allCapabilitiesAvailable(t) \equiv$
    $\forall rc \in requiredCaps(t),\ \exists c \in nodeCapabilities(self) match(rc, c) \wedge taskAssigned(self, c) = undef$
  $taskIsBeingProcessed(node, task) \equiv taskProcessingMode(self, t) \neq undef$
```

```
**InitTask**$(t) \equiv$
  **if** $requiredCaps(t) = undef$ **then**
    $requiredCaps(t) \leftarrow$ ComputeCapabilities$(t)$
```

```
**AssignTaskToNode**$(task, node) \equiv$
  $assignedChildNode(self, task) := node$
  SendMessage$(node, newTaskMassage(task))$
  $taskMode(task) := assigned$
  **remove** $task$ **from** $taskPool(self)$
**DecomposeTask**$(task) \equiv$
  **forall** $t' \in taskDecomposition(task)$ **do**
    $parentTask(t') := task$
    **add** $t'$ **to** $taskPool(self)$
    **add** $t'$ **to** $subTasks(task)$
    $taskMode(t') := assigned$
```

In order to do decomposition, we break the task into subtasks and put all of them in our task pool.

5.2.3 Matching Child Node

FindMatchingChildNode rule which is depicted in Figure 5.6 tries to find an appropriate node for the required task. The diagram provides the overview of the process and its control flow, the details of the conditions and the actions they trigger are defined in terms of an ASM. For instance, the following predicates specifies the condition '*Is there a candidate?*':

$$candidateExists(node, task) \equiv \exists\ c \in childNodes(node) \quad isACandidate(c, task)$$
$$isACandidate(node, task) \equiv \forall\ rc \in requiredCapabilities(task) \quad quote(node, task)(rc) \neq undef$$

and the following rules define the actions 'Send Quote Requests to Child Nodes' and 'Receive Quotes' of the resource management (RM) Layer:

```
**SendQuoteRequests**$_{RM}(task) \equiv$
  **forall** $c$ **in** $activeChildNodes(self)$ **do**
    **let** $m = new(\textsc{Message})$ **in**
      $msgType(m) := quoteRequest$
      $msgData(m,$ "task"$) := task$
      $msgData(m,$ "capabilities"$) := requiredCapabilities(task)$
      $quote(c, task) := undef$
      SendMessage$_{RM}(c, m)$
where
  $activeChildNodes(n) = \{x \mid x \in childNodes(n) \land \neg isDead(x)\}$
```

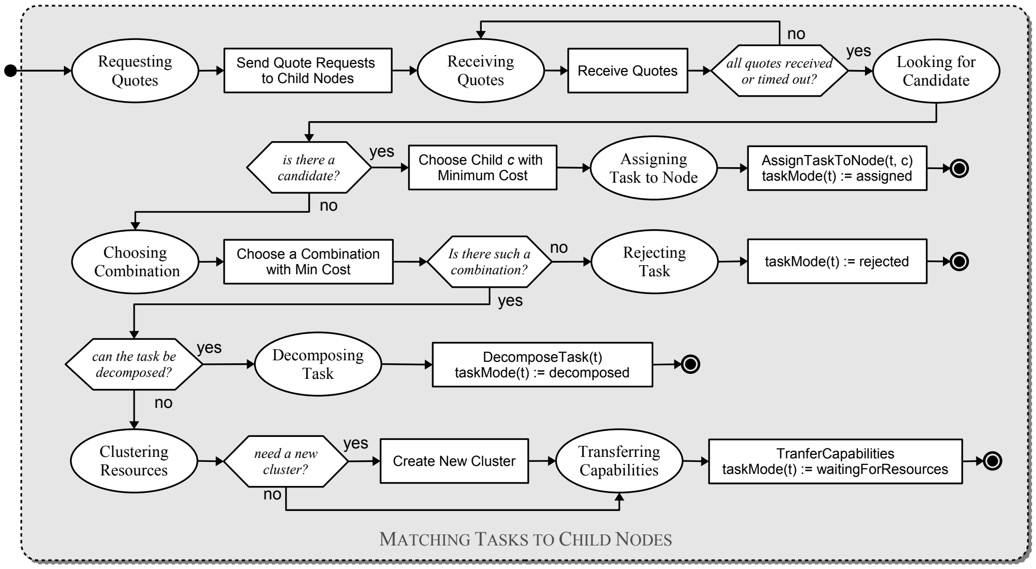

Requesting Quotes
Send Quote Requests to Child Nodes
Receiving Quotes
Receive Quotes
all quotes received or timed out?
no
yes
Looking for Candidate
is there a candidate?
yes
no
Choose Child *c* with Minimum Cost
Assigning Task to Node
AssignTaskToNode(t, c) taskMode(t) := assigned
Choosing Combination
Choose a Combination with Min Cost
Is there such a combination?
no
yes
Rejecting Task
taskMode(t) := rejected
can the task be decomposed?
yes
no
Decomposing Task
DecomposeTask(t) taskMode(t) := decomposed
Clustering Resources
need a new cluster?
yes
no
Create New Cluster
Transferring Capabilities
TranferCapabilities taskMode(t) := waitingForResources
MATCHING TASKS TO CHILD NODES

Figure 5.6: Control state diagram of matching tasks to child nodes

```
$\textsf{\textbf{ReceiveQuotes}}_{RM}(task) \equiv$
  choose $m \in inbox(self)$ with
      $msgType(m) = quoteResponse \wedge msgData(m, \text{“task”}) = task$ do
    $quote(sender(m), task) := msgData(m, \text{“quote”})$
    remove $m$ from $inbox(self)$
```

The above rules exhibit two important abstractions in the model. In these rules, the abstract view of communication services and data structures of messages allows us to focus on the main functionality of the process. To send quote requests to child nodes, the model relies on the messaging services provided by the communication layer. For every child node, a new *quote request* message is created asking for a quote on the cost of performing the new task, and the message is sent using the abstract routine $\textsf{SendMessage}_{RM}$. In the next step, the node looks into its message inbox and non-deterministically chooses quote messages related to the new task and stores the received quotes in an internal data structure to be used later in the process (see ReceiveQuotes above). This rule is repeated until all the expected quotes have either been received or timed-out.

If no single resource cluster suitable to perform the new task is available, the best combination of resources from different clusters are selected to form a new cluster (see “Choosing Combination” in Figure 5.6). To create a new cluster, a new logical resource and a corresponding node is created, and the hierarchical structure is then modified by changing the values of functions *parentNode* and *childNodes*, effectively adding the new cluster node to the tree (see $\mathsf{CreateNewClusterNode}_{LD}$ above). The following rules define the action 'Create New Cluster' of the resource management layer and the respective rule it uses from the logical distribution (LD) layer:

```
**CreateNewCluster**$_{RM}$ $\equiv$
  $newClusterNode(self) \leftarrow \mathsf{CreateNewClusterNode}_{LD}$

**CreateNewClusterNode**$_{LD}$ $\equiv$
  **let** $nr = new(\textsc{Resource})$ **in**
    $\mathsf{ConfigureResource}(nr, emptyCluster, noCapability)$
    $\mathsf{CreateAndConfigureNode}(nr)$
    **add** $node(nr)$ **to** $childNodes(self)$
    $parent(node(nr)) := self$
    **result** $:= node(nr)$
```

The next step is to transfer the selected resources of the winning combination into the newly created cluster. Here, the resource manager layer directly uses the transfer capabilities service provided by the logical distribution layer (see $\mathsf{TransferCapabilities}_{LD}$ below). Based on the selected combination, a transfer map is created. Transfer of resources is done using a messaging protocol: for every pair of $(node, capability)$ in the given transfer map, a *transfer request* message is sent to *node* requesting to transfer its *capability* to the new cluster. Resources acknowledge a successful transfer of their capabilities by sending back a *transfer ack* message, which is collected in the next step.

The following rule defines the action 'Transfer Capabilities' of the logical distribution (LD) layer. Here we provide a simple implementation of the protocol. The node keeps a set of the capability transfer requests that it has sent so far, and then it removes those requests for which an acknowledgment is received. The transfer is considered to be successful when all the requests are acknowledged before a time-out event occurs.

```
**TransferCapabilities**$_{LD}(task, target, transferMap) \equiv$
    **if** $requestsSent = undef$ **then**
        $outcome := undef$
        $timer \leftarrow$ GetNewTimer
        SendCapabilityTransferRequests$_{LD}(task, target, transferMap)$
    **else**
        **if** $|requestsSent| > 0$ **then**
            **if** $timedOut(timer)$ **then**
                $outcome := failed$
            **else**
                RemoveProcessedTransferRequests$(self, task, target, transferMap)$
        **else**
            $requestsSent := undef$
            $outcome := successful$
where
    $requestsSent = capabilityTransferRequestsSent(self, task, target, transferMap)$
    $outcome = transferResult(self, task, target, transferMap) := successful$
    $timer = transferTimer(self, task, target, transferMap)$
```

```
**SendCapabilityTransferRequests**$_{LD}(task, target, transferMap) \equiv$
    **seq**
        $requestsSent := \{\}$
    **next**
        **forall** $(node, cap)$ **in** $transferMap$ **do**
            **extend** MESSAGE **with** $m$ **do**
                $msgType(m) := transferRequest$
                $msgData(m,$ "capability"$) := cap$
                $msgData(m,$ "task"$) := task$
                $msgData(m,$ "target_node"$) := target$
                SendMessage$(node, m)$
                **add** $(node, cap)$ **to** $requestsSent$
where
    $requestsSent = capabilityTransferRequestsSent(self, task, target, transferMap)$
```

5.3 Simple Scenario

In this section, we present a simple scenario modeled with CoreASM[6]. We assume that we have a network of resources (Figure 5.7 illustrates the corresponding tree structure of the network, in which each node represents a physical or logical resource) with *Base* as the root node. In this scenario, there are two logical nodes under *Base*: *GroupA* and *GroupB*; each of these nodes has a set of physical or logical resources as its child nodes. There are different number of physical resources of types (*aurora*, *helicopter*, *boat* and *frigate*) in the system. As you can see in the figure, there are two nodes, *Helicopter-1* and *Helicopter-2*, with resource type *helicopter*. The capabilities of these resources can be defined as follows:

$$helicopter\begin{cases} mobility: & \langle \textit{type, `air'}\rangle, \langle \textit{speed, `high'}\rangle \\ vision: & \langle \textit{type, `day'} \text{ and } \textit{`night'}\rangle, \langle \textit{quality, `high'}\rangle \\ rescue: & \langle \textit{type, `lift'}\rangle \end{cases}$$

$$boat\begin{cases} mobility: & \langle \textit{type, `sea'}\rangle, \langle \textit{speed, `medium'}\rangle \\ vision: & \langle \textit{type, `day'}\rangle, \langle \textit{quality, `medium'}\rangle \\ rescue: & \langle \textit{type, `tow'}\rangle \end{cases}$$

$$aurora\begin{cases} mobility: & \langle \textit{type, `air'}\rangle, \langle \textit{speed, `high'}\rangle \\ vision: & \langle \textit{type, `day'}\rangle, \langle \textit{quality, `low'}\rangle \end{cases}$$

$$frigate\begin{cases} mobility: & \langle \textit{type, `sea'}\rangle, \langle \textit{speed, `medium'}\rangle \\ vision: & \langle \textit{type, `day'} \text{ and } \textit{`night'}\rangle, \langle \textit{quality, `high'}\rangle \end{cases}$$

At the outset, a *Patrol* task is assigned to *Base*. Processing its tasks pool, the *Base* computes the required capabilities for the given task in order to find proper resources (child node) for the task assignment. As explained in Section 5.1.4, the required capabilities for a task are defined by a set of *Capability Patterns*. For instance, in this scenario, the required capabilities needed for *Patrol* task are defined as follows:

$$Patrol\begin{cases} mobility: & \langle \textit{type, `sea'} \text{ OR } \textit{`air'}\rangle, \langle \textit{speed, `> low'}\rangle \\ vision: & \langle \textit{type, `day'}\rangle, \langle \textit{quality, `medium'}\rangle \end{cases}$$

In this case, *helicopter*, *boat* and *frigate* are all capable of performing the task. However, it is more cost efficient to have *Boat* performing this task; hence, *Base* will assign the task to *GroupB*. Upon receiving the task, *GroupB* queries its child nodes for cost estimations

[6]The scenario presented here is aligned with the view of the system explained in this chapter.

and realizes that *Boat* is the best candidate to perform the task.

While *Boat* is patrolling the area, suppose a new *SOS* task is assigned to *Base*. The required capabilities for this task are as follows:

$$SOS \begin{cases} \textit{mobility}: & \langle \textit{type, `sea'} \text{ AND } \textit{`air'}\rangle, \langle \textit{speed, `> medium'}\rangle \\ \textit{vision}: & \langle \textit{type, `day'} \text{ AND } \textit{`night'}\rangle, \langle \textit{quality, `high'}\rangle \\ \textit{rescue}: & \langle \textit{type, `lift'} \text{ AND } \textit{`tow'}\rangle \end{cases}$$

The capabilities needed for this task can not be found in any of *GroupA* or *GroupB* exclusively. Thus, *Base* determines that it cannot assign the task to a single child node, therefore, the task must be decomposed. For this example we can assume that *SOS* task can only be broken down into two subtasks of *Search* and *Rescue* with the following capability patterns:

$$Search \begin{cases} \textit{mobility}: & \langle \textit{type, `sea'} \text{ AND } \textit{`air'}\rangle, \langle \textit{speed, `> medium'}\rangle \\ \textit{vision}: & \langle \textit{type, `day'} \text{ AND } \textit{`night'}\rangle, \langle \textit{quality, `high'}\rangle \end{cases}$$

$$Rescue \begin{cases} \textit{mobility}: & \langle \textit{type, `sea'} \text{ AND } \textit{`air'}\rangle, \langle \textit{speed, `> medium'}\rangle \\ \textit{vision}: & \langle \textit{type, `day'} \text{ AND } \textit{`night'}\rangle, \langle \textit{quality, `medium'}\rangle \\ \textit{rescue}: & \langle \textit{type, `lift'} \text{ AND } \textit{`tow'}\rangle \end{cases}$$

As a result, *helicopter* and *frigate* are both needed to carry out *Search* task. Therefore, *Base* will assign the task to *GroupA*. Since *Search* task consists of two different searches in the area, *GroupA* will further decompose the *Search* and assigns *Search_Air* and *Search_Sea* to *Helicopter-2* and *Frigate* respectively. Whenever a task is decomposed into subtasks, the original task stays in the node where it was decomposed (the task mode should be changed to *decomposed*) until all its subtasks are completed or at least one of them is failed. In our example, when the sinking boat is found by any of the resources (*Frigate* or *Helicopter-2*) deployed to the area, the task mode of both subtasks will be changed to *Completed* and a report containing the status of victims and the boat will be sent to the parent node. Subsequently, information is bubbled up through the hierarchy, so that *Base* is notified about the completion of *Search* task, hence triggering *Rescue* task into the system.

When *Rescue* task is introduced by *Base* node, a similar procedure is carried out to find the matching child node for the task. Assuming that the capabilities needed for *Rescue* operation are ‘lift’ and ‘tow’, where the victims should be lifted up and the boat needs to be towed to a secure location, both *boat* and *helicopter* resources are needed to carry out this task. Therefore, *Rescue* task should be assigned to *GroupB* and later to *GroupC* in

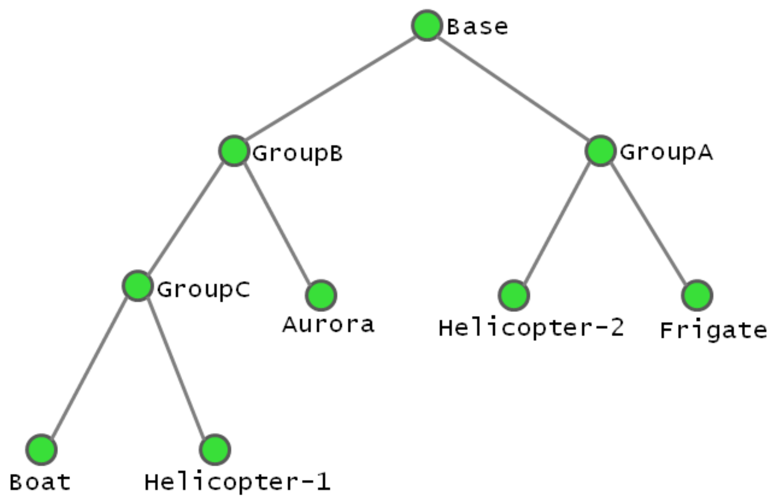


Figure 5.7: Structure of the resource hierarchy

the hierarchy. After decomposition to subtasks *Lift* and *Tow*, *GroupC* realizes that *Boat* is busy with *Patrol* task. However, since rescue has a higher priority than patrolling, the node suspends *Patrol* operation and sets the task mode to *Suspended* (it will stay suspended until all the required resources become available again). By suspending the low-priority task, both *Helicopter-1* and *Boat* will be available and *Rescue* operation can be performed.

Upon the completion of *Rescue* operation (i.e. victims are rescued and the boat is secured on the shore), the task mode is switched to *Completed*, effectively releasing the resources. At this point, the suspended task (*Patrol*), which was residing in the task pool of *Boat* node, can be resumed for processing.[7]

Finally, *SOS* task mode will be determined by the success or failure of *Rescue* operation, and the report is sent back to the commander in charge.

[7]When the mission is not accomplished successfully, the task will be terminated with *Rejected* mode. Consequently, the parent node tries to reassign the task to other resources (if possible).

Chapter 6

Experimental Analysis

Generally, evaluation of a system can be based on various criteria. Each of these criteria focuses on certain aspects of the system. It is also important to take into account the domain of the system, when choosing appropriate metrics. For any complex decision support system, validity of the results is one of the main factors that should be tested even in early stages of design. The need for validation is even more highlighted when the system is operating in a dynamically changing environment. Such an additional requirement raises another important issue to consider in system evaluation which is referred to as the robustness of the system. In other words, robustness is the key requirement for any real time decision support system. Therefore, in our application context where the system deals with real life situations such as emergency responses, validity and robustness are the two most important metrics which place other criteria such as efficiency or cost in second priority.

In the previous chapters, we have proposed an abstract model of the system. Since the model is formally described with Abstract State Machine formalism, its specifications can be implemented at different levels of abstraction by using the CoreASM tool environment in order to evaluate the model before building the whole system. In this chapter, our goal is to provide a number of test schemes in order to validate the results of running different scenarios in the system. The scenarios are designed in a way to resemble dynamic changes happening in the real system in order to better analyze the robustness of the system.

It is important to note that measuring robustness in the system is not a trivial issue. Although there is no well-established quantitative metrics to evaluate the robustness of the system; intuitively, robustness refers to the situation where the system works validly in the presence of dynamic changes, still operating reasonably well when confronted with sudden failure of resources.

In this chapter we analyze the robustness of our system based on the validity of the results defined in terms of correct task assignments to available resources under various circumstances. In order to do so, we have extended the scope of scenarios introduced in the previous chapter and run the system on a series of such scenarios, in which the dynamic changes in the system are modeled by random failure of resources. In the next step, the results obtained from system runs will be compared to our expected results, and differences will be analyzed based on Error Type I and II. The outcome of such analysis will give us an insight into the robustness of our system.

6.1 Evaluation metrics

As mentioned earlier in the thesis, our decision support system is meant to provide help in decision making processes regarding the assignment of tasks based on the availability of resources with matching capabilities. Therefore, evaluation of task assignment helps us to assess the validity of the system. Each task assignment is a result of the system run with a series of inputs (e.g. scenarios). For any given task in a scenario, the system's response can be either negative—rejection of the task—or positive—successful assignment of one or more matching resources to a given task. Moreover, in order to simulate dynamic changes in the system, scenarios are chosen in a way that resources get disabled non-deterministically at different times.

The results of the system will be evaluated by comparing them to the standard results obtained from a reference model. The results of our reference model are gained by a brute-force algorithm that explores the problem space to generate solutions based on the knowledge about the common concepts of Marine Safety and Security Operations. This knowledge is acquired through close collaborations with the MSOC experts familiar with the domain. This reference model acts as a standard benchmark for the evaluation of our model, also known as *gold standard* [74]. Apparently, acquiring such results is very costly yet needed to provide a baseline to analyze the validity of our system. In other words, if a result obtained from our model matches the standard result, it will be considered as *true* otherwise it will be *false*. Therefore, each result from the system falls into one of the four cases depicted in Table 6.1. As shown in the table, there are two types of errors for any given result.

		Gold Standard	
		Positive	**Negative**
Test Results	**Positive**	Correct outcome True Positive	Type I error False Positive
	Negative	Type II error False Negative	Correct outcome True Negative

Table 6.1: Types of test results

- **Type I errors:** Also known as an error of the first kind, refers to a *false positive* result in which the system has a positive result for the given question while it in not true in reality. An example of this error can be seen in diagnosis systems, where a positive result shows a disease when in fact the patient doesn't have any, or when a security scanner in the airport detects a dangerous thing in a passenger's luggage while there's no such a thing in there. Type I error can be seen in our model as the system response is positive for the given task assignment, while the configuration of resources chosen for such task can not satisfy the task required capabilities. In this situation, the system shows wrong answers.

- **Type II error**: This type of error, also known as a β error or a *false negative*, is the error of failing to response positively, meaning that the answer to a given question is true in reality while the results show opposite. To understand this type of error, we can refer to the same diagnosis system; one can consider the situation where the result shows nothing for a patient who in fact suffers from a disease, and similarly, when a metal detector in airport fails to detect a person with a weapon. In our model, a false negative result refers to the situation in which the system cannot find any set of resources for a task assignment, while considering the capability of resources and current configuration of the network, it is possible to do so; for instance, different scheduling techniques or re-planning can be used to perform the introduced task.

  In statistics, type I errors are also known as α errors, whereas type II is called β. These parameters are defined as follows:

$$\alpha = \frac{FP}{FP\ +\ TN}, \quad \beta = \frac{FN}{FN +\ TP}$$

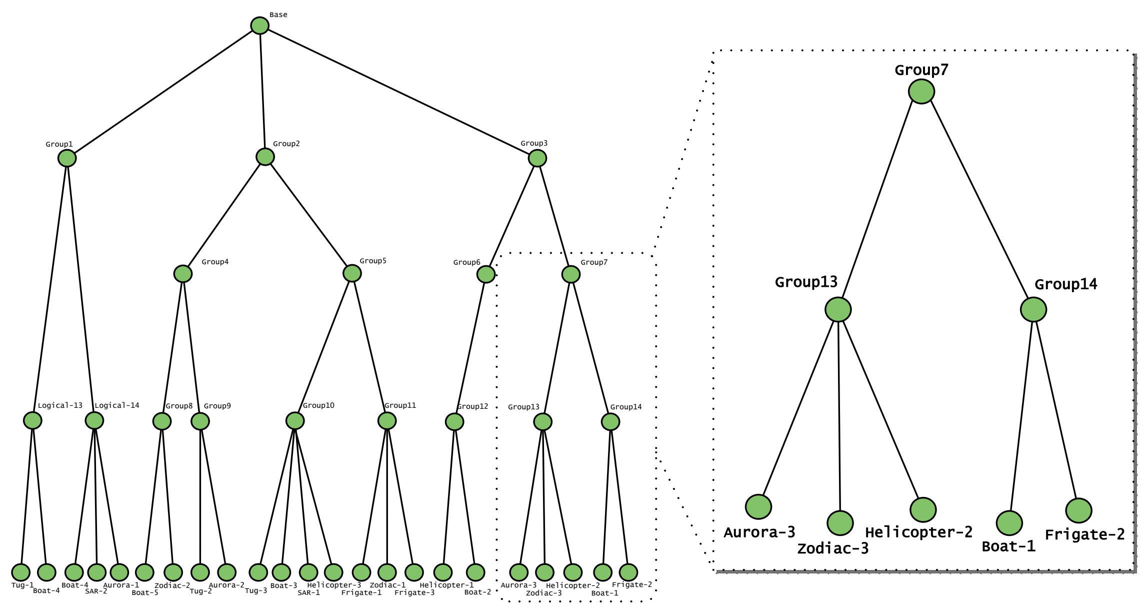


Figure 6.1: Resource network for scenario testing

In addition, *sensitivity* is described as a conditional probability of getting true positive results from a test while the true answer is positive. This can be defined as $1 - \beta$ and is evaluated as the proportion of *True Positive* results by the total of positive values. By this definition, it can be easily recognized that when a system is highly sensitive, it means that the probability of getting false negative results are very low. For instance, a highly sensitive diagnosis system barely ignores any symptom of disease.

Similarly, *specificity* is referred to the ability of a test to provide true negative results, which is defined as $1 - \beta$ or in other words, the ratio of *True Negative* results to the total of negative values. Likewise, when the diagnosis system is specific, the chance of recognizing a disease for a healthy patient is very unlikely.

6.2 Test Setup

We test the system by running a number of different scenarios. Each scenario consists of a list of tasks, which are introduced to the system at specific times, and an initial configuration of resources in the system. These scenarios are very similar to what is described in Section 5.3. However, our chosen scenarios are more complex in terms of task requirements and configuration of resources in the system (see Figure 6.1).

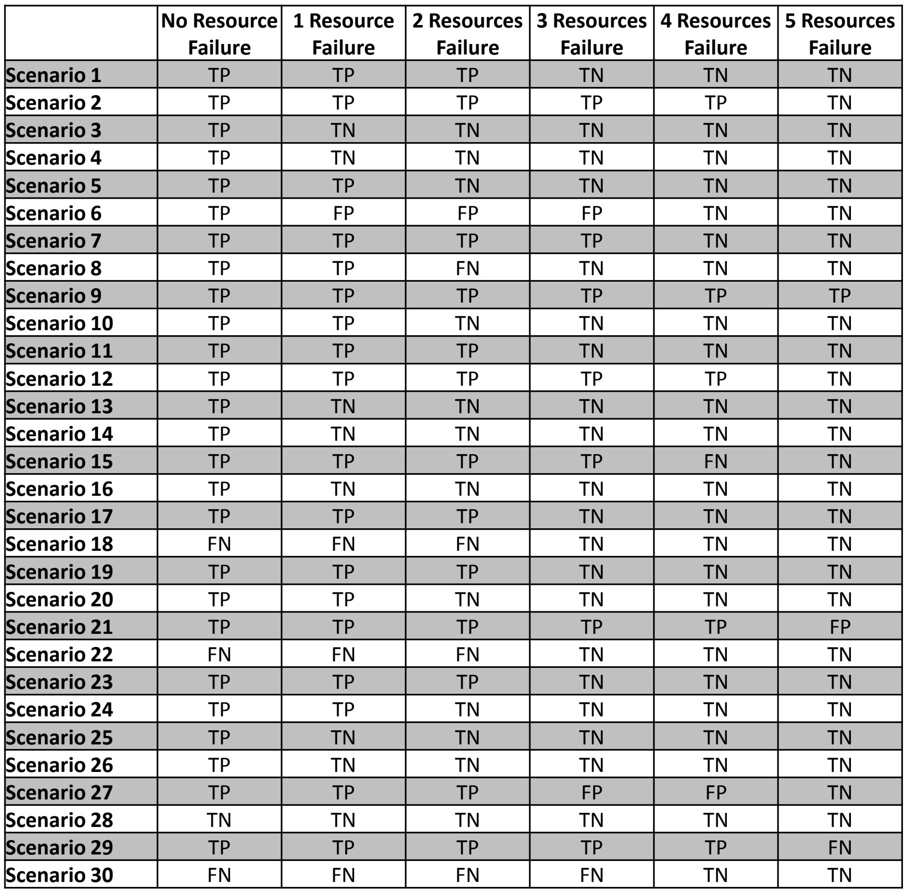
Our data set includes 30 scenarios that are frequently encountered in daily routine of the Marine Safety and Security Operation domain. Furthermore, to resemble dynamic changes in the system, we randomly disable some of the resources that are busy performing tasks.[1] We observe how the system copes with changes in the environment and analyze the results of tasks assignments by comparing them to our standard benchmarks, in order to associate their values with truth or falseness. We summarize the results in a table where the value of the element in row i and column j corresponds to the result of running scenario i with random failure of j resources and can take up one of the four possible values mentioned before in Table 6.1. In addition, since the failure of resources has a random nature, we run each scenario 10 times to get possibly different responses from the system, hence obtaining 10 such tables.

6.3 Simulation Results

Table 6.2 shows the results of one run of the scenarios for different number of resource failures. First column represents the results of running scenarios in a normal situation with no sudden changes in the availabilities of resources, while other columns show how the system behaves in the presence of progressive resource failures—one or more resources die. In addition, for each column of the table, sensitivity $(1 - \beta)$ corresponds to the rate of true positive results and α represents the rate of false positive results. Therefore, to better understand and compare the results, Receiver Operative Characteristic (ROC) diagram [77] is used (see Figure 6.3). In this diagram, sensitivity is plotted against α for every column of all tables. In the digram, scenario runs with the same number of resource failures (RF) are shown with the same symbol. Conceptually, each point in the figure can be interpreted as the results of running 30 scenarios under the specified number of resource failures, and the whole digram can be used to measure the validity of the system with respect to changes in the environment.

The ROC diagram in Figure 6.3 shows that the rates of false positive results (α values) are very small (less than 0.2). This shows that the system barely assign tasks to the resources

[1]To better analyze how the system responds to changes, one of the busy assigned resources is selected non-deterministically at each step. We repeat this procedure until we reach a total failure of 5 resources at once.



	No Resource Failure	1 Resource Failure	2 Resources Failure	3 Resources Failure	4 Resources Failure	5 Resources Failure
Scenario 1	TP	TP	TP	TN	TN	TN
Scenario 2	TP	TP	TP	TP	TP	TN
Scenario 3	TP	TN	TN	TN	TN	TN
Scenario 4	TP	TN	TN	TN	TN	TN
Scenario 5	TP	TP	TN	TN	TN	TN
Scenario 6	TP	FP	FP	FP	TN	TN
Scenario 7	TP	TP	TP	TP	TN	TN
Scenario 8	TP	TP	FN	TN	TN	TN
Scenario 9	TP	TP	TP	TP	TP	TP
Scenario 10	TP	TP	TN	TN	TN	TN
Scenario 11	TP	TP	TP	TN	TN	TN
Scenario 12	TP	TP	TP	TP	TP	TN
Scenario 13	TP	TN	TN	TN	TN	TN
Scenario 14	TP	TN	TN	TN	TN	TN
Scenario 15	TP	TP	TP	TP	FN	TN
Scenario 16	TP	TN	TN	TN	TN	TN
Scenario 17	TP	TP	TP	TN	TN	TN
Scenario 18	FN	FN	FN	TN	TN	TN
Scenario 19	TP	TP	TP	TN	TN	TN
Scenario 20	TP	TP	TN	TN	TN	TN
Scenario 21	TP	TP	TP	TP	TP	FP
Scenario 22	FN	FN	FN	TN	TN	TN
Scenario 23	TP	TP	TP	TN	TN	TN
Scenario 24	TP	TP	TN	TN	TN	TN
Scenario 25	TP	TN	TN	TN	TN	TN
Scenario 26	TP	TN	TN	TN	TN	TN
Scenario 27	TP	TP	TP	FP	FP	TN
Scenario 28	TN	TN	TN	TN	TN	TN
Scenario 29	TP	TP	TP	TP	TP	FN
Scenario 30	FN	FN	FN	FN	TN	TN

Figure 6.2: Results of scenarios runs

which are not capable of performing them. The false positive results mainly occur with the sudden failure of two or more resources. The concurrent nature of the system along with the dynamic changes, results in system's failure to notice the death of a resource in a timely manner.

Moreover, the distribution of points in the diagram confirms the high rates of true positive results (sensitivity values), which infers low rates of false negatives ($sensitivity = 1 - \beta$). It was also noted that these false negative results are either the consequence of static decomposition of tasks or the system's inability to perform re-planning when faced with dynamic changes in the availability of resources. The system produced false negative results more often in the situations with smaller numbers of failed resources, as the more

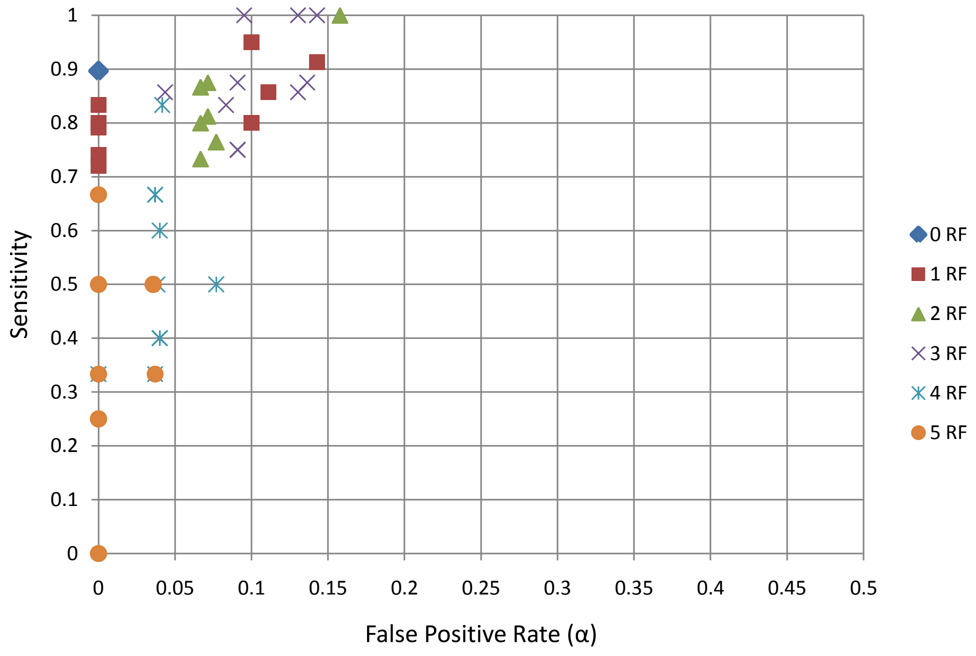


Figure 6.3: Receiver Operating Characteristic diagram[2]

resources were failed, the higher was the chance of true negative results.

6.4 Conclusions

In conclusion, in order to measure the robustness of our system, we examined the validity of the system with a certain number of scenarios under dynamic changes in the availability of resources. The promising results showed that the system is resilient enough to cope with unpredictable changes. As to further improve the results, we can enrich the system by providing dynamic re-planning during any stages of task execution to reduce the possibility of having false negative results. Moreover, information fusion techniques can be exploited to increase situation awareness and prevent any false results caused by outdated knowledge

[2]Scenario runs with a certain number of resource failures are shown with the same symbol on the diagram. RF refers to the resource failures.

about current state of the system. Such features are critical for emergency response situations with stringent time constraints where any failure in task assignments can highly reduce the functionality of the system.

It's worth mentioning that although there are various other metrics to study the behavior of the system, the focus of our analysis was on the robustness, which is essential for any system working in a highly dynamic environment. Moreover, measuring other metrics requires implementation of the system in a lower abstract level with more details which is beyond the work presented in this thesis.

Chapter 7

Conclusions

Distributed information fusion and information sharing across a heterogeneous distributed network (e.g., consisting of surveillance assets or search and rescue platforms) operating in an adverse, dynamic and uncertain environment require an intelligent, robust and scalable framework to integrate the different views in a coherent and consistent model. Resource Configuration Management and Task Management are the two central and coupled issues that are critical for the ability to adapt to dynamic changes in the distributed information fusion typology, the availability and the conditions of its constituents.

The process of Dynamic Resource Configuration Management for surveillance platforms can be described in terms of dynamic management of distributed, heterogeneous and autonomous resources. The key challenge of this process, is to dynamically manage the above described resources in order to optimize the achievement of the mission goals and to ensure the adaptability to a continuously changing operating environment and network. In addition, Decision Support can be recognized as the process of Dynamic Task Management, where the main focus is on the resource allocation for given tasks, considering the current state of the system, resource availability, task constraints, and priorities. Moreover, by taking into account the uncertainty in the system, caused by internal or external changes in an unpredictable environment, the robustness of the system can be maintained by introducing dynamic re-planning. Both aspects, resource configuration management and task management, are challenging problems that require reliable and accurate models to systematically analyze, reason about and compare possible solutions.

The work presented here aims at proposing a comprehensive formal framework that facilitates not only the integration between dynamic resource configuration management and dynamic task management into an integrated model, but also allows focusing on and modeling the interplay between the two aspects. Arguably, there is no reliable way of developing any comprehensive system concept without having the ability to perform quantitative experiments. Specifically, the concurrent and reactive behaviour of the underlying algorithms and protocols make it difficult to predict the resulting properties accurately. This situation calls for quantitative analysis and experimental validation of design decisions in early design phases, prior to actually building the system. To address this issue, we have used CoreASM as a platform for experimental validation. Early stage prototypes of the system along with the evaluation of the results gained after various scenario runs showed satisfying results for the stakeholders.

Future Work

In future we will work on the Planning component in a way that the current situation of the system have more impact on the plan generation. In other words, a complex mission introduced to the system can be decomposed into different combination of subtasks, hence generating different plans. Our goal will be to find the optimal solution to the decomposition problem, based on the resource availability, task prioritization, load balancing, and etc.

As mentioned in Chapter 4, the need for a dynamic re-planning is vital for a system operating in a dynamically changing environment. Therefore, the Planning component should be enriched in a way that re-planning can take place in a real-time manner in any steps of task execution.

Another issue in our proposed model which needs further improvements is scheduling of the resources for given tasks. By taking into account, time constraint, availability of resources in future, load balancing, and etc., an scheduler can be attached to the system as a new component, which will facilitate the resource allocation process.

Appendix A

ASM Formalism

This section outlines the formal modeling framework at an intuitive level of understanding using common notions and structures from computational logic and discrete mathematics. For details, we refer to the existing literature on the theory of abstract state machines [47, 65].

Abstract State Machines [15] are known for their versatility in semantic modeling of algorithms, architectures, languages, protocols and virtually all kinds of sequential, parallel and distributed systems. Widely recognized applications include semantic foundations of popular industrial system design languages, like SDL [39], VHDL [13] and SystemC [60] (the ASM model of SDL is part of ITU's SDL standard [52]), programming languages, like JAVA [71] and C# [12], Web services [32], communication architectures [41], embedded control systems [14], wireless networks [40], among many others.[1] Leaning towards practical applications of formal methods, a driving factor for the development of ASM specification, validation and verification techniques employed in the above applications has been the desire to systematically reveal abstract architectural and behavioural concepts inevitably present in every system design, but often not in an explicit form, so that the underlying semantic blueprint of the functional system requirements becomes clearly visible and can be inspected and checked by analytical means.

[1]Information on ASM applications and foundations is provided by the ASM Research Centre at www.asmcenter.org.

Concurrency, Reactivity and Time

The asynchronous computation model of Distributed ASM (DASM) defines concurrent and reactive behaviour, as observable in distributed computations performed by autonomously operating computational agents, according to the underlying semantic model in terms of *partially ordered runs* [41].

A DASM M is defined over a given vocabulary V by its program P_M and a non-empty set I_M of initial states. V consists of a finite collection of symbols denoting mathematical objects and their relation in the formal representation of M, where we distinguish *domain symbols*, *function symbols* and *predicate symbols*. Symbols that have a fixed interpretation regardless of the state of M are called *static*; those that may have different interpretations in different states of M are called *dynamic*. A state S of M results from a valid interpretation of all the symbols in V and constitutes a variant of a first-order structure, one in which all relations are formally represented as Boolean-valued functions.

Concurrent control threads in an execution of P_M are modeled by a dynamic set AGENT of computational *agents*. This set may change dynamically over runs of M, as required to model a varying number of computational resources. Agents of M interact with one another, and possibly also with the operational environment of M, by reading and writing shared locations of a global machine state. The underlying semantic model regulates such interactions so that potential conflicts are resolved according to the definition of partially ordered runs.

P_M consists of a statically defined collection of agent programs $P_{M_1}, ..., P_{M_k}$, $k \geq 1$, each of which defines the behaviour of a certain *type* of agent in terms of state transition rules. The canonical rule consists of a basic update instruction of the form

$$f(t_1, t_2, ..., t_n) := t_0,$$

where f is an n-ary dynamic function symbol and the $t_i{}'s$ $(0 \leq i \leq n)$ are terms. An update instruction specifies a pointwise function update, i.e., an operation that replaces an existing function value by a new value to be associated with the given function arguments. Complex rules are inductively defined by a number of well defined rule constructors allowing the composition of rules for describing sophisticated behavioural patterns.

A computation of an individual agent of M, executing program P_{M_j}, results in a finite or infinite sequence of state transitions of the form

$$S_0 \overset{\Delta_{S_0}(P_{M_j})}{\longrightarrow} S_1 \overset{\Delta_{S_1}(P_{M_j})}{\longrightarrow} S_2 \overset{\Delta_{S_2}(P_{M_j})}{\longrightarrow} \cdots ,$$

such that S_{i+1} is obtained from S_i, for $i \geq 0$, by firing $\Delta_{S_i}(P_{M_j})$ on S_i, where $\Delta_{S_i}(P_{M_j})$ denotes a finite set of updates computed by evaluating P_{M_j} over S_i. Firing an update set means that all the updates in this set are fired simultaneously in one atomic step. The result of firing an update set is defined if and only if the set does not contain any conflicting updates (attempting to assign different values to the same location).

A DASM M interacts with a given operational environment—the part of the external world visible to M—through actions and events as observable at external interfaces, formally represented by externally controlled functions. Intuitively, such functions are manipulated by the external world rather than the agents of M. Of particular interest are *monitored functions*. Such functions change their values dynamically over runs of M, although they cannot be updated internally by agents of M. A typical example is the abstract representation of global system time. In a given state S of M, the global time (as measured by some external clock) is given by a monitored nullary function *now*, taking values in a linearly ordered domain TIME. Values of *now* increase monotonic over runs of M. Additionally, $'\infty'$ represents a distinguished value of TIME, such that $t < \infty$ for all $t \in \mathsf{TIME} \setminus \{\infty\}$. Finite time intervals are given as elements of a linearly ordered domain DURATION.

Bibliography

[1] Natural Resources Canada, The Atlas of Canada – Coastline and Shoreline. Last visited Mar 2010.

[2] Royal canadian mounted police, marine security operation centres (msoc), 2010. Last visited, February 2011.

[3] J.S. Albus. The engineering of mind. In *From animals to animats 4: proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, volume 4, page 23. The MIT Press, 1996.

[4] F. Amato, M. Fiorini, S. Gallone, and G. Golino. Fully solid state radar for vessel traffic services. In *Radar Symposium (IRS), 2010 11th International*, pages 1–5. IEEE.

[5] D. Amyot. Introduction to the user requirements notation: learning by example. *Computer Networks*, 42(3):285–301, 2003.

[6] P. Arabie, L.J. Hubert, and G. De Soete. *Clustering and classification.* World Scientific Pub Co Inc, 1996.

[7] Ali Khalili Araghi. Net-Enabled Adaptive Distributed Information Fusion for Large Volume Surveillance (NADIF). Last visited June 2010.

[8] N.F. Ayan, U. Kuter, F. Yaman, and R.P. Goldman. Hotride: Hierarchical ordered task replanning in dynamic environments. In *Proceedings of the 3rd Workshop on Planning and Plan Execution for Real-World Systems (held in conjunction with ICAPS 2007).* Citeseer, 2007.

[9] G. Beni and J. Wang. Swarm Intelligence in Cellular Robotic Systems. 1989.

[10] E. Börger. The ASM ground model method as a foundation of requirements engineering. *Verification: theory and practice: essays delivered to Zohar Manna on the occasion of his 64th birthday*, page 145, 2003.

[11] E. Börger. The Abstract State Machines method for high-level system design and analysis. *Formal Methods: State of the Art and New Directions*, pages 79–116, 2010.

[12] E. Börger, N. G. Fruja, V. Gervasi, and R. F. Stärk. A High-level Modular Definition of the Semantics of C#. *Theoretical Computer Science*, 336(2/3):235–284, May 2005.

[13] E. Börger, U. Glässer, and W. Müller. Formal Definition of an Abstract VHDL'93 Simulator by EA-Machines. In C. Delgado Kloos and P. T. Breuer, editors, *Formal Semantics for VHDL*, pages 107–139. Kluwer Academic Publishers, 1995.

[14] E. Börger, E. Riccobene, and J. Schmid. Capturing Requirements by Abstract State Machines: The Light Control Case Study. *Journal of Universal Computer Science*, 6(7):597–620, 2000.

[15] E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.

[16] É. Bossé, J. Roy, and S. Ward. *Models and Tools for Information Fusion*. 2007.

[17] S. Breban and J. Vassileva. Long-term coalitions for the electronic marketplace. In *Proceedings of Canadian AI Workshop on Novel E-Commerce Applications of Agents*, pages 6–12. Citeseer, 2001.

[18] Shifrin C.A. Gate assignment expert system reduces delays at United's hubs. *Aviation Week & Space Technology*, 128(4):148–159, 1988.

[19] S. Camazine. *Self-organization in biological systems*. Princeton University Press, 2001.

[20] O. Dobrican. An example of collaborative system. In *International Workshop Collaborative Support Systems in Business and Education, Risoprint, Cluj-Napoca*, page 48, 2005.

[21] J.E. Doran, S. Franklin, N.R. Jennings, and T.J. Norman. On cooperation in multi-agent systems. *The Knowledge Engineering Review*, 12(3):309–314, 1997.

[22] M. Dorigo and T. Stützle. *Ant colony optimization*. MIT press, 2004.

[23] Turban E. and Aronson J. E. *Decision Support Systems and Intelligent Systems. 6th edition*. Prentice Hall, 6th edition, 2001.

[24] M. Egerstedt and X. Hu. Formation constrained multi-agent control. *Robotics and Automation, IEEE Transactions on*, 17(6):947–951, 2001.

[25] Mica R. Endsley. Toward a theory of situation awareness in dynamic systems. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, pages 32–64, 1995.

[26] K. Erol. Semantics for hierarchical task-network planning. Technical report, MARYLAND UNIV COLLEGE PARK INST FOR SYSTEMS RESEARCH, 1995.

[27] BS Everitt, S. Landau, and M. Leese. Cluster analysis. 2001. *Arnold, London*, 2001.

[28] R. Farahbod, V. Gervasi, and U. Glässer. CoreASM: An Extensible ASM Execution Engine. *Fundamenta Informaticae*, pages 71–103, 2007.

[29] R. Farahbod and U. Glässer. Dynamic Resource Management for Adaptive Distributed Information Fusion in Large Volume Surveillance—Phase One. Technical Report SFU-CMPT-TR-2008-08, Simon Fraser University, April 2008.

[30] R. Farahbod, U. Glässer, and A. Khalili. A Multi-Layer Network Architecture for Dynamic Resource Configuration & Management of Multiple Mobile Resources in Maritime Surveillance. In *Proc. of SPIE Defense & Security Symposium*, March 2009.

[31] R. Farahbod, U. Glässer, A. Khalili, and Adel Guitouni. Dynamic Resource Management for Adaptive Distributed Information Fusion in Large Volume Surveillance—Phase Two. Technical Report SFU-CMPT-TR-2009-05, Simon Fraser University, March 2009.

[32] R. Farahbod, U. Glässer, and M. Vajihollahi. An Abstract Machine Architecture for Web Service Based Business Process Management. *International Journal of Business Process Integration and Management*, 1:279–291, 2007.

[33] R. Farahbod, U. Glässer, and H. Wehn. Dynamic Resource Management for Adaptive Distributed Information Fusion in Large Volume Surveillance. In *Proc. of SPIE Defense & Security Symposium*, March 2008.

[34] Roozbeh Farahbod and Vincenzo Gervasi. JASMine: Accessing Java Code from CoreASM. In *Proceedings of the Dagstuhl Seminar on Rigorous Methods for Software Construction and Analysis (LNCS Festschrift)*, 2008. (to be published).

[35] J. Farley. *Java distributed computing.* O'Reilly Media, 1998.

[36] S. Franklin and A. Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. *Intelligent Agents III Agent Theories, Architectures, and Languages*, pages 21–35, 1997.

[37] Dan Galorath. Software Project Failure Costs Billions-Better Estimation & Planning Can Help, June 2008.

[38] M. Ghallab, D.S. Nau, and P. Traverso. *Automated Planning: theory and practice.* Morgan Kaufmann Publishers, 2004.

[39] U. Glässer, R. Gotzhein, and A. Prinz. The Formal Semantics of SDL-2000: Status and Perspectives. *Computer Networks*, 42(3):343–358, 2003.

[40] U. Glässer and Q.-P. Gu. Formal Description and Analysis of a Distributed Location Service for Mobile Ad Hoc Networks. *Theoretical Comp. Sci.*, 336:285–309, May 2005.

[41] U. Glässer, Y. Gurevich, and M. Veanes. Abstract Communication Model for Distributed Systems. *IEEE Trans. on Soft. Eng.*, 30(7):458–472, July 2004.

[42] U. Glässer, P. Jackson, A. Araghi, H. Wehn, and H. Shahir. A collaborative decision support model for marine safety and security operations. *Distributed, Parallel and Biologically Inspired Systems*, pages 266–277, 2010.

[43] U. Glässer, P. Jackson, A. Khalili Araghi, and H. Yaghoubi Shahir. Intelligent Decision Support for Marine Safety and Security Operations. In *Intelligence and Security Informatics, IEEE International Conference on Intelligence and Security Informatics, ISI 2010, Vancouver, Canada*, Lecture Notes in Computer Science. Springer, 2010.

[44] D.N. Godbole and J. Lygeros. Longitudinal control of the lead car of a platoon. *Vehicular Technology, IEEE Transactions on*, 43(4):1125–1135, 1994.

[45] A.D. Gordon. Hierarchical classification. *Clustering and classification*, pages 65–121, 1996.

[46] S. Guerlain, D.E. Brown, and C. Mastrangelo. Intelligent decision support systems. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 3, pages 1934–1938. IEEE, 2000.

[47] Y. Gurevich. Sequential Abstract State Machines Capture Sequential Algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111, July 2000.

[48] B. Horling and V. Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(04):281–316, 2004.

[49] IALA. *VTS Manual*, 4th edition.

[50] O. Ilghami. Documentation for jshop2. *Department of Computer Science, University of Maryland, Tech. Rep*, 2006.

[51] P. Ireland, R. Case, J. Fallis, C. Van Dyke, J. Kuehn, and M. Meketon. The Canadian Pacific Railway transforms operations by using models to develop its operating plans. *Interfaces*, pages 5–14, 2004.

[52] ITU-T Recommendation Z.100 Annex F (11/00). *SDL Formal Semantics Definition.* International Telecommunication Union, 2001.

[53] J.P. Kahan and A. Rapoport. *Theories of coalition formation.* Lawrence Erlbaum, 1984.

[54] J. Kennedy. Swarm intelligence. *Handbook of Nature-Inspired and Innovative Computing*, pages 187–219, 2006.

[55] M. Klusch and A. Gerber. Dynamic coalition formation among rational agents. *Intelligent Systems, IEEE*, 17(3):42–47, 2002.

[56] M. Mauve, J. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad-hoc networks. *IEEE Network*, 15(6), 2001.

[57] C. McCann and R. Pigeau. Clarifying the Concepts of Control and of Command. In *Proceedings of the 1999 Command and Control Research and Technology Symposium*, pages 475–490, 1999.

[58] C. Merida-Campos and S. Willmott. Modelling coalition formation over time for iterative coalition games. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 572–579. IEEE Computer Society, 2004.

[59] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system: A toolkit for building multi-agent simulations, 1996.

[60] W. Müller, J. Ruf, and W. Rosenstiel. An ASM Based SystemC Simulation Semantics. In W. Müller et al., editors, *SystemC - Methodologies and Applications.* Kluwer Academic Publishers, June 2003.

[61] G. Mussbacher and D. Amyot. Goal and scenario modeling, analysis, and transformation with jucmnav. In *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 431–432. IEEE.

[62] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[63] Gregor Pavlin, Marinus Maris, and Jan Nunnink. An agent-based approach to distributed data and information fusion. In *IAT*, pages 466–470, 2004.

[64] ITU-T Recommendations. Z.150-z.159: User requirements notation. 2009.

[65] W. Reisig. On gurevich's theorem on sequential algorithms. *Acta Informatica*, 39(4):273–305, 2003.

[66] S.J. Russell and P. Norvig. Artificial intelligence: a modern approach. 2003.

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

[68] M. Sims, C.V. Goldman, and V. Lesser. Self-organization through bottom-up coalition formation. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 867–874. ACM, 2003.

[69] L.K. Soh, C. Tsatsoulis, and H. Sevay. A satisficing, negotiated, and learning coalition formation architecture. *Distributed sensor networks: a multiagent perspective*, pages 109–138, 2003.

[70] J. Staff. Joint Publication 1-02: Department of Defense Dictionary of Military and Associated Terms. *Washington, DC*, 2001.

[71] R. Stärk, J. Schmid, and E. Börger. *Java and the Java Virtual Machine: Definition, Verification, Validation.* Springer-Verlag, 2001.

[72] CCW Team. CanCoastWatch System Concept. Technical report, Technical report, MacDonald, Dettwiler and Associates Ltd., 2006.

[73] R. Vahidov and B. Fazlollahi. Pluralistic multi-agent decision support system: a framework and an empirical test. *Information & Management*, 41(7):883–898, 2004.

[74] S. Wacholder, B. Armstrong, and P. Hartge. Validation studies using an alloyed gold standard. *American journal of epidemiology*, 137(11):1251, 1993.

[75] H. Wehn et al. A Distributed Information Fusion Testbed for Coastal Surveillance. In *Proc. of the 10th Intl. Conf. on Information Fusion*, July 2007.

[76] M. Wooldridge and N. Jennings. Agent theories, architectures, and languages: a survey. *Intelligent agents*, pages 1–39, 1995.

[77] X.H. Zhou, N.A. Obuchowski, and D.K. McClish. *Statistical methods in diagnostic medicine*, volume 414. LibreDigital, 2002.

Glossary

AIS	Automatic Identification System
ASM	Abstract State Machine
C2	Command and Control
DASM	Distributed Abstract State Machine
DRCMA	Dynamic Resource Configuration Management Architecture
DSS	Decision Support System
HTN	Hierarchical Task Network
IA	Intelligent Agent
ITU-T	International Telecommunication Union - Telecommunication Standardization Sector
MAS	Multi-Agent Systems
MDA	MacDonald, Dettwiler and Associates Ltd.
MSOC	Marine Security Operation Centre
NADIF	Net-Enabled Adaptive Distributed Information Fusion
RCMP	Royal Canadian Mounted Police
RMA	Resource Management Architecture
ROC	Receiver Operating Characteristic
SA	Situation Awareness
SDL	Specification and Description Language
STL	Software Technology Lab
TM	Task Management
URN	User Requirements Notation
VTS	Vessel Traffic Service