In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

The undersigned hereby recommend to the Faculty of Graduate Studies and Research the
acceptance of the thesis:

Distributed Mutual Exclusion in Cooperating Mobile Agents

Submitted By

Fayyaz Ahmad

A handwritten signature in black ink, reading "Douglas Howe". The signature is written in a cursive, flowing style.

Professor Douglas Howe

(Director, School of Computer Science)

**Professor Nicola Santoro
(Thesis Supervisor)**

**Dr. Amiya Nayak
(Thesis Co-Supervisor)**

Carleton University
August 28, 2003

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitions et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-88764-2

Our file Notre référence

ISBN: 0-612-88764-2

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Canada

ABSTRACT

Mobile agents represent a promising technology for the development of distributed applications. Most of these applications consist of multiple agents that need to interact with one another at various stages of their execution for exchanging information and synchronization. The coordination and cooperation of multiple agents require that the traditional problem of distributed mutual exclusion be addressed in the context of mobile agent environment. This work identifies that the available traditional solutions may not be suitable for mobile agent paradigm due to the factors of mobility and openness of the network, contrary to the static processes and limited network in the client server distributed systems. There is also a need to design solutions that exploit the computational power and benefits of mobility provided by mobile agents. This dissertation presents a platform independent, distributed, scalable and fault tolerant solution to the problem of distributed mutual exclusion in a mobile agent system.

LIST OF TABLES

Table 3.1 – System Agents and Responsibilities.....	33
Table 4.1 - Response Time & Agent Migrations Under Heavy Load (10-Hosts).....	67
Table 4.2 - Response Time & Agent Migrations Under Light Load (10-Hosts).....	69
Table 4.3 - Response Time & Agent Migrations Under Heavy Load (5-Hosts).....	71
Table 4.4 - Response Time & Agent Migrations Under Light Load (5-Hosts).....	73

LIST OF FIGURES

Table 3.1 – System Agents and Responsibilities	33
Figure 3.1 - Symbols for Agent Representation	38
Figure 3.2 – Initialization of Directory.....	39
Figure 3.3 – Request for Mutual Exclusion.....	40
Figure 3.4 – Granting mutual Exclusion.....	41
Figure 3.5 - Simple Example (1)	42
Figure 3.6 - Simple Example (2)	43
Figure 3.7 - Simple Example (3)	43
Figure 3.8 - Simple Example (4)	44
Figure 3.9 - Concurrent Example (1).....	45
Figure 3.10 - Concurrent Example (2)	45
Figure 3.11 - Concurrent Example (3)	46
Figure 3.12 - Concurrent Example (4)	46
Figure 3.13 - Rule-1	49
Figure 3.14 – Rule-2.....	49
Figure 4.1 - Response Time Under Heavy Load (10-Hosts)	68
Figure 4.2 - Average Agent Migrations Under Heavy Load (10-Hosts).....	68
Figure 4.3 - Response Time Under Light Load (10-Hosts)	70
Figure 4.4 - Average Agent Migrations Under Light Load (10-Hosts)	70
Figure 4.5 - Response Time Under Heavy Load (5-Hosts).....	72
Figure 4.6 - Average Agent Migrations Under Heavy Load (5-Hosts)	72
Figure 4.7 - Response Time Under Light Load (5-Hosts).....	74
Figure 4.8 - Average Agent Migrations Under Light Load (5-Hosts)	74

ACKNOWLEDGMENTS

I am highly thankful to Dr. Nicola Santoro and Dr. Amiya Nayak for their guidance and supervision. Their help and assistance made it possible to finish this work. My friends at Carleton University deserve special thanks for their help and encouragement.

I am grateful to my brothers and sisters. Their love and care have been a motive force for all of my endeavors. I am thankful to my wife for her patience and encouragement when it was most required. I am forever indebted to my parents for their efforts, support and prayers.

TABLE OF CONTENTS

INTRODUCTION	3
1.1 Introduction.....	3
1.2 Problem Statement	4
1.3 Goals and Objectives	5
1.4 Related Work	5
1.5 Thesis Contribution.....	6
1.6 Thesis Organization.....	7
BACKGROUND	8
2.1 Distributed Systems.....	8
2.1.1 Advantages of Distributed Systems	8
2.1.2 Distributed Computing Models.....	9
2.2 Software Agents	11
2.3 Classification of Software Agents	12
2.4 Mobile Software Agents.....	13
2.5 Mobile Agent Standardization	13
2.6 Pros and Cons of Mobile agent Paradigm	14
2.6.1 Advantages of Mobile Agents.....	14
2.6.2 Limitations of Mobile Agents	16
2.7 Application of Mobile Agents.....	16
2.8 Coordination Models in mobile Agents	18
2.9 Mobile Agent Systems	19
2.9.1 Aglets	19
2.9.2 TACOMA.....	20

2.9.3 Voyager.....	20
2.9.4 Odessy.....	21
2.9.5 Concordia.....	22
2.9.6 Mole.....	23
2.10 Distributed Mutual Exclusion.....	24
2.11 Survey of Solutions to Distributed Mutual Exclusion.....	25
2.12 Distributed Mutual Exclusion in Mobile Agent Environment	27
SOLUTION	30
3.1 Overview	30
3.2 Assumptions	36
3.2.1 General Assumptions	36
3.2.2 Fault Assumptions.....	37
3.3 Description of the Solution	38
3.3.1 Initialization.....	38
3.3.2 Request	39
3.3.3 Grant	40
3.3.4 Release.....	41
3.4 Examples of Solution.....	41
3.4.1 Simple Example.....	42
3.4.2 Concurrent Example.....	44
3.5 Fault Tolerance	47
3.5.1 Definitions	48
3.5.1 Rules for Fault Tolerance.....	48
3.5.2 Fault Scenarios	50
3.6 Mutual Exclusion Solution Pseudo Code.....	53
3.6.1 Worker Agent Pseudo Code.....	53
3.6.2 Locator Agent Pseudo-Code	54
3.6.3 Guide Agent Pseudo Code	55
3.6.4 Mutual Exclusion Agent Pseudo Code	56
3.7 Correctness	57
3.8 Cost Analysis	59
3.9 Solution Properties	60
3.9.1 Distributed Solution	60

3.9.2 Decoupling.....	60
3.9.3 Network Organization Independence	61
3.9.4 Fault Tolerance	61
3.9.5 Protocol Encapsulation.....	61
PROTOTYPE IMPLEMENTATION AND TESTS	62
4.1 Objectives and Scope.....	62
4.2 Environment Selection	63
4.3 ASDK Overview.....	63
4.4 Implementation Details	65
4.5 Test Methodology	65
4.6 Test Results.....	66
4.6.1 Test Case 1: 10-Nodes Under Heavy Load Conditions	67
4.6.2 Test Case 2: 10-Nodes Under Light Load Conditions	69
4.6.3 Test Case 3: 5-Nodes Under Heavy Load Conditions	71
4.6.4 Test Case 4: 5-Nodes Under Light Load Conditions	73
4.7 Analysis of Test Results	75
CONCLUSIONS AND FUTURE WORK.....	76
5.1 Thesis Summary	76
5.2 Conclusions.....	77
5.3 Future work.....	78
APPENDIX-I	80
APPENDIX-II	97

Chapter 1

INTRODUCTION

1.1 Introduction

Mobile agents represent a promising technology for the development of distributed applications [2, 3, 6]. The key idea in this model is that in many cases it is more efficient and convenient to move a client code to server nodes and have it executed on the server nodes where it has local access to the resources needed. With increasing acceptance of the mobile agent based computing model, more and more complex computations are being designed and implemented using this paradigm. A large number of these applications consist of multiple agents that need to interact with one another at various stages of their execution for exchanging information and synchronization.

The coordination and cooperation of multiple agents bring the traditional problems, like mutual exclusion, deadlock avoidance and leader election, under discussion in the mobile agent based systems. Although a myriad of solutions is available to these classical problems, they could not be applied directly to agent based systems. The most significant reason is the factor of mobility and openness of the network in this paradigm, contrary to the static processes and limited network in the client server distributed systems. The fundamental building blocks of many traditional distributed solutions rely on assumptions such as data location, communication mechanism or network organization, which may not be valid for mobile agent based execution environments. More over traditional solutions may not exploit the

computational power and benefits of mobility gained from mobile agents. There is a need to look into above mentioned coordination problems with a focus on mobile agent based execution environments. New solutions must be developed to address these coordination problems effectively and efficiently in mobile agent based systems.

1.2 Problem Statement

Mutual exclusion is one of the most fundamental problems in computing systems. The problem of mutual exclusion becomes much more complex in distributed systems, and many algorithms have been proposed in distributed systems where nodes communicate by asynchronous message passing [7]. This problem stems from the need of coordination among different nodes to complete a global task.

It is evident that mobile agent based systems have to deal with the problem of distributed mutual exclusion to provide efficient solutions for distributed computing problems. Coordination is required for applications consisting of teams of mobile agents, where each agent is responsible for performing part of a common, global task. Teams of mobile agents are likely to become the means to implement several distributed and network applications in future [8].

The available solutions for distributed mutual exclusion may not work in the mobile agent based computing paradigm due to the complications introduced by the agent movement and openness of the network. Mobile agent based systems impose their own restrictions and provide new powers of computing. It is necessary to design solutions that leverage the powers of this new dynamic paradigm while following its limitations.

1.3 Goals and Objectives

The main objective of the thesis is to design a solution to the distributed mutual exclusion problem among cooperating mobile agents. The solution suits the new requirements of mobile agents and aims to fully exploit its powers. Following are the main objectives and goals of the presented solution.

The proposed solution should not impose any unnecessary restrictions to the agility of mobile agents. It should meet the requirements and challenges of mobile agent systems. Centralized solutions suffer from problems of scalability and locality. Therefore the solution presented by this dissertation must be non-centralized or distributed. It should not rely on any platform specific services so that it is not tightly coupled to any particular mobile agent environment. It should be easy and simple to implement and integrate this solution in any mobile agent environment that provides basic functionality of inter agent communication. The presented solution should consider fault tolerance aspects and must provide mechanism to handle different fault scenarios.

1.4 Related Work

Mobile agent is an emerging technology and a lot of work has been done in the design and implementation of execution environment, but until now only few environments provide higher level services that give support for the coordination, in general, and for distributed mutual exclusion, in particular, among cooperating mobile agents.

One work in this area is by Vera Nagamuta and Markus Endler [10]. They propose a coordination mechanism for cooperating groups of mobile agents and demonstrate its use to

achieve mutual exclusion. Their mechanism is based on message diffusion or broadcast. Moreover they introduced a concept of fixed domains for groups of cooperating agents and restricted the inter domain migration of agents. This mechanism imposes an overhead of groups and a fixed proxy for a group of agents.

S. Mishra and P. Xie [6] designed and implemented synchronization mechanism for DaAgent System. They use a centralized approach to provide location independent coordination and synchronization. A process, called “Clearing House” is chosen to be the coordination place. This solution is specifically designed for DaAgents and so is implementation dependent. It uses simple and easy to implement home-based approach. This is a centralized approach and, under heavy loads, “Clearing House “ is likely to become a synchronization bottleneck. In general, home-based schemes suffer from problems of scalability and locality.

Jiannong Cao, Xianbing Wang, and Jie Wu [8] presented a fully distributed algorithm using mobile agents to achieve mutual exclusion in a networking environment. This algorithm is designed within a specific framework for mobile agent enabled distributed server groups. This work is about solving the old problem of distributed mutual exclusion, among nodes or servers, using mobile agents. It does not address the problem of mutual exclusion among the participating mobile agents

1.5 Thesis Contribution

This work presents a study of the problem of distributed mutual exclusion among cooperating mobile agents. The main contribution of the thesis is to present a distributed, scalable and fault tolerant solution to the problem of distributed mutual exclusion in a mobile agent based

system. A prototype of the solution is also implemented on a mobile agent platform, IBM's Aglet, one of the most known environments for mobile agents. Test results and analysis is also included to measure the performance of presented solution.

1.6 Thesis Organization

This thesis is divided into five chapters. Chapter two provides a general overview of mutual exclusion problem, available solutions and problems in application of these solutions to mobile agent environment. This chapter also provides an introduction to mobile agents, their coordination models and their properties. In Chapter three, we propose a distributed, scalable and fault-tolerant solution to the problem of mutual exclusion among cooperating mobile agents. This chapter also provides examples and analysis of the presented solution. Chapter four contains the details of prototype implementation, test methodology, test results and analysis. In Chapter five, we draw conclusions of the thesis work and suggest guidelines for future research.

Chapter 2

BACKGROUND

In this chapter, we provide a general overview and history of distributed computing, mobile agents and problem of distributed mutual exclusion. First we introduce different paradigms of distributed computing. It follows a survey of a specific distributed computing paradigm; mobile agents. Then we describe problem of distributed mutual exclusion and a survey of available traditional solutions. Finally mobile agents systems are compared to traditional distributed environment in the context of distributed mutual exclusion problem.

2.1 Distributed Systems

A distributed system consists of a collection of autonomous computers linked together by a network; the network is equipped with distributed system software that enables computers to coordinate activities and to share the resources of the system [5]. It is an environment where we can harness idle CPU cycles and storage space of tens, hundreds, or thousands of networked systems to work together on a particularly processing-intensive problem [11]. Distributed computing systems allow processing to be done using the method of divide and conquer [12].

2.1.1 Advantages of Distributed Systems

Distributed computing is an integral part of modern computing systems. Following are few key factors responsible for usefulness of distributed computing [5].

- **Resource sharing:** Distributed computing allows an organization to make the best use of the available physical resources.
- **Efficiency:** Multiple platforms in a heterogeneous environment can be utilized so that each computer has to solve only part of the problem.
- **Scalability:** Since distributed systems are built out of multiple components, it is possible to add more components as needed. Distributed systems may be used to solve problems of any size since they scale with the problem.
- **Fault tolerance and availability:** Components of a distributed application can be configured to survive most type of failures.
- **Transparency:** The system is perceived as a whole rather than as collection of independent components. Therefore local and remote objects can be accessed using identical operations, with out knowledge of their location.

2.1.2 Distributed Computing Models

Distributed computing models can be classified using different criteria such as architecture, process communication, network topology etc. Architecture and process communication based computing models are more relevant to this dissertation. In this section, we discuss these computing models in detail.

2.1.2.1 Architecture Models

There are two main categories of the architecture models: client/server model and mobile agent model [4,5].

1. Client/Server Model

In this model, a server advertises a set of services that provide access to some resources. The server hosts the code that implements these services locally. So it is the server that executes the code and has the processing power. If a client is interested in accessing the resource hosted by the server, the client will simply use one or more of the services provided by the server. So far, most distributed systems have been based on this paradigm. Some examples are remote procedure calling, object request brokers (CORBA) and java remote method invocation (RMI).

2. Mobile Agent Model

In this model there is no concept of static client and servers. Instead code for services is encapsulated in mobile entities and these entities are capable of moving to any node. This exciting paradigm has many interesting properties that we discuss later in this chapter.

2.1.2.2 Communication Models

Distributed computing models can also be classified by the mechanism employed for inter-process communications [34]. Following are the main types of process models.

1. Message Passing Models

In this model, processes communicate by message passing mechanism. A process sends a message by adding it to a message queue, and another process receives the message by removing it from the queue. This model is further divided depending upon factors like the length of the message queues and how long a delay may occur between when a message is sent and when it can be received.

2. Shared Variable Models

In this model, processes communicate through global shared variables. These variables can be read and written by all processes. A more restrictive class of this model permits inter process communication only through local shared variables.

3. Synchronous Communication Models

In this mode, each process must respond, to a call from any other process, without initiating a new communication. Unlike the case of ordinary message passing the input and output commands are executed synchronously. This model was introduced by Hoare in his work, Communicating Sequential Processes (CSP) language [35].

2.2 Software Agents

There is no single definition available for the term software agent or agent [13]. Agents are supposed to exhibit some fundamental characteristics. For this thesis we consider the following properties of an agent [4,18].

An agent is a software object that

- Is situated within an execution environment.
- Possesses the following mandatory properties:
 - Reactive: senses changes in the environment and acts according to those changes.
 - Autonomous: has control over its own actions.

- Goal-driven: is proactive.
- Temporally continuous: is continuously executing.
- And may possess any of the following orthogonal properties:
 - Communicative: able to communicate with other agents.
 - Mobile: can travel from one host to another.
 - Learning: adapts in accordance with previous experience.
 - Believable: appears believable to the end user.
 - Flexible: not scripted.

2.3 Classification of Software Agents

There are several dimensions to classify existing software agents. The classification that interests this thesis is based upon the ability of agents to move around some network. This yields the classes of static or stationary and mobile agents.

A stationary agent executes only on the system where it begins its execution. If it needs information that is not on that system or needs to interact with another agent, it typically uses a communication mechanism. In contrast, a mobile agent is not bound to the system where begins its execution. A mobile agent is free to travel among the host in the network.

For the rest of the thesis we are concerned with mobile agents only. The term agent means mobile agent unless stated otherwise.

2.4 Mobile Software Agents

There is no single definition of a mobile agent. Since this is an active area of research, we can find a myriad of formal definitions. The following definition describes the characteristics of a mobile agent in the context of this thesis.

A mobile agent is a program that can migrate from host to host in a network of heterogeneous computer systems and fulfill a task specified by its owner. It works autonomously and communicates with other agents and host systems. During the self-initiated migration, the agent carries all its code and the complete execution state with it.

The mobility of an agent could be strong or weak. Strong mobility is found when data, code and state are transferred between two hosts. On the other hand, weak mobility involves the transfer of data, or data and code. The different types of mobility have their advantages and disadvantages, but the agents used in this thesis have the strong form of mobility.

2.5 Mobile Agent Standardization

Several organizations promote standards for mobile agent systems. Some of these organizations are Object Management Group (OMG), Agent Society and Foundation for Intelligent and Physical Agents (FIPA). The most significant effort Mobile Agent System Interoperability Facility (MASIF), so far, was presented by few important vendors and accepted by OMG.

MASIF addresses the interfaces between agent systems written in same language. MASIF standardize the following four areas.

- Agent management
- Agent transfer
- Agent and agent system names
- Agent System type and location syntax

2.6 Pros and Cons of Mobile agent Paradigm

Mobile agent paradigm provides some interesting properties to make it very useful for some applications. Like any other method it also has its limitations and disadvantages. In this section we cover the benefits and challenges of this exciting field of distributed computing.

2.6.1 Advantages of Mobile Agents

Following is a list of advantages gained by the mobile agent paradigm [4,14,15,16,17].

- **Network Load Reduction:** This means that computations are moved to data and less information is transferred across the network.
- **Network Latency Avoidance:** Network latency is reduced through centralized agent dispatch and local execution of user commands.
- **Asynchronous and autonomous execution:** Continually open network connections are not required to control mobile agents.

- **Adaptability and flexibility:** Since mobile agents sense and react to their environment, they are inherently adaptable and flexible.
- **Heterogeneous network tolerance:** Mobile agents are separated from platform specific issues; therefore they can operate in variety of environments.
- **Protocol encapsulation:** Agents contain all information related to a protocol and allows easy upgrading and implementation of proprietary protocols.
- **Robust and fault-tolerant execution:** The dynamic adaptability of agents allows them to function in less than perfect conditions. For example if a node fails, a mobile agent can leave the node and continue to execute elsewhere.
- **Overcoming the limitations of a client computer:** Issues such as limited memory, computing power and storage space can be avoided by moving the mobile agent to a more suitable node.
- **Customization:** Searches and queries can be adjusted by user criteria or via the mobile agent detecting properties specific to remote site.
- **Ease of developing and analogy to the real world:** Complex programming problems are often easier to understand when related to the physical world.

2.6.2 Limitations of Mobile Agents

Mobile agents provide an innovative way for distributed computing, but there are certain limitations associated with it. Most of these limitations stem from the infancy stage of this paradigm. Following is a list of disadvantages presented by the researchers.

- **Lack of standards:** There have been standardization efforts, as described above in section 2.4. But these standards are not fully complete and deployed. The lack of standards makes the development costs high.
- **Security:** This is an active area of research. Improvements are being made to achieve safer environments, but malicious agents can be harmful to the host or to the network.
- **Lack of infrastructure:** Being a new technology, it is a difficult and time consuming to make use different agent systems in collaboration or to embed mobile agents into other popular technologies such as web browsers.
- **Lack of applications:** This is the most common disadvantage of mobile agent systems. There are many mobile agent implementations but very few applications.

2.7 Application of Mobile Agents

Despite the fact that there are not many computing problems that cannot be solved without mobile agents, mobile agents can make certain applications easier to develop and may improve efficiency and reliability. Scholars who have worked in this area have pointed out many applications that would benefit from this new paradigm. Joachim Baumann [5] has listed few interesting applications as:

- **Electronic Commerce:** It is certainly one of the most attractive application areas for mobile agents. A commercial transaction may require real-time access to remote resources. A mobile agent can act and negotiate on behalf of its user in buying, selling or trading goods, services or information.
- **Distributed Information Retrieval:** Instead of moving large amounts of data through the network to extract the needed information on the client side, an agent can be sent to remote information sources, where the information can be extracted locally.
- **Monitoring:** An agent can be sent to monitor a given information source or wait for a specific event. As soon as a change occurs, the agent reacts according to its programmed behavior.
- **Workflow Management Systems:** Mobile agents can provide mobility, behavior, information about the workflow and autonomy to every workflow item.
- **Information Dissemination:** Mobile agents can disseminate information to a number of customers. It is very easy to control different access policies and output format for the information.
- **Parallel Processing:** Mobile agents can distribute parallel processes very easily. The mobile agents distributing the computation might even redistribute themselves to adapt to changes in the environment.

- **Software Deployment:** Mobile agents can be used to automate the process of software installation and software updates. They can query the target node to gather information about its environment and preferences and transfer the suitable package.

2.8 Coordination Models in mobile Agents

Coordination is required for applications consisting of teams of mobile agents, where each agent is responsible for performing part of a common, global task. Teams of mobile agents are likely to become the means to implement several distributed and network applications in future [9].

Many coordination models have been proposed for mobile agents. Cabri et al [26] has defined four categories of coordination models based on spatial and temporal coupling.

- **Direct Coordination:** In this model mobile agents send messages directly to each other. This model imposes both spatial (because the sender must know the receiver's identity) and temporal (because the receiver must be active during the communication) coupling.
- **Meeting-Oriented Coordination:** In this model, mobile agents interact through meeting points (a place where meeting can take place). It limits the spatial coupling by removing the necessity of location and name sharing before communication.
- **Blackboard-based Coordination:** In this model, the agents interact through the shared message repositories at each place, called blackboards. It imposes spatial coupling. The main advantage of this model is that it is free of temporal coupling.

- **Linda-like Coordination:** This model is also based on shared name-space, but unlike blackboards, it uses an associative tuples space where information is organized as tuples and can be accessed through pattern matching. It provides both temporal and spatial uncoupling.

The coordination and communication services provided by the agent platform are important for the solution this thesis proposes. Although we implement our solution using a specific implementation, to demonstrate the working, it is not platform specific. We rely on the very basic direct communication between agents. Almost all agent platforms we investigate do provide this basic communication service. Platform specific, more efficient solutions may be devised to exploit other coordination models, but this is beyond the scope of this thesis.

2.9 Mobile Agent Systems

The mobile agent based model is developing dynamically and so fast that we cannot cover all the systems available. This section presents a description of the most known and popular agent platforms. Since IBM's Aglet is selected for the implementation of our proposed solution, it is covered in detail later in chapter-5. The selection of a particular system should not influence the ability of a developer to implement our proposed solution for mutual exclusion among mobile agents.

2.9.1 Aglets

Aglet [28] models the mobile agent to closely follow the applet model of Java. It is a simple framework where the programmer overrides predefined methods to add desired functionality. An Aglet is defined as a mobile Java object that visits Aglet-enabled hosts in a computer

network. Aglet runs in its own thread of execution after arriving at the host, so it is attributed as autonomous. It is also reactive because it responds to incoming messages. The complete Aglet object model includes additional abstractions such as context, proxy, message, itinerary, and identifier. These additional abstractions provide Aglet the environment in which it can carry out its tasks. Aglet uses a simple proxy object to relay messages and has a message class to encapsulate message exchange between agents. However, group-oriented communication is not available, and the choice of using a proxy to relay a message may not be a scalable solution in a high-frequency transport situation. Nevertheless, by modeling the mobile agent as a Java object, the designers of Aglet leverage the existing Java infrastructure to take care of platform dependent issues and to use existing mobile code facility of Java.

2.9.2 TACOMA

The TACOMA [29] (Tromsø and Cornell Moving Agents) project represents an early attempt to build a mobile agent system. In TACOMA the agent is modeled as a migrating process that moves through the network to satisfy client requests. The TACOMA project focuses on operating system support for mobile agent and how mobile agent can help solve problems traditionally addressed by operating systems. The system itself is written in C. The system supports agents written in C, Tcl/Tk, Perl, Python and Scheme (Elk). Overall, it is unclear how far this approach can progress, since TACOMA is the only project that conducts investigation on implementing mobile agent at the operating system level.

2.9.3 Voyager

ObjectSpace's [30] Voyager is a platform for agent-enhanced distributed computing in Java. The Voyager agent model is based on the concept of a collection of Java objects; it does have

an Agent class that developers can subclass to implement Voyager-style mobile agent. However, the product goes one step beyond to provide arbitrary remote object construction, a facility for moving objects (not just agents), and a host of other communication and infrastructure services that can be used to implement arbitrary mobile agent systems. Voyager allows java programmers to create network applications using both traditional and agent-enhanced distributed programming techniques.

2.9.4 Odessa

The Odyssey [33] project shares (or rather inherits) many features from a previous General Magic product: Telescript. However, the amount of open documentation on the Odyssey system is rather terse; therefore, its description will be limited. The Odyssey MA model also centers on a collection of Java objects, more similar in concept to Aglet than to Concordia or Mole. The top-level classes of the Odyssey system are Agent, Worker, and Place. Worker is a subclass of Agent and represents an example of what a developer can do with the Agent class. An Odyssey Place class is an abstraction of where an Odyssey agent exists and performs work. A special facility such as directory service is associated with Place. Odyssey agents communicate using simple method calls, and Odyssey does not support high-level communication. However, Odyssey agents can form and destroy meeting places to exchange messages. There is also an undocumented feature regarding global communication to a "published" object, but this feature is not officially supported. The distinctive feature of Odyssey is its design to accommodate multiple transport mechanisms. Currently, Odyssey supports Java Remote Method Invocation (RMI), Microsoft Distributed Component Object Model (DCOM), and CORBA Internet Inter-ORB Protocol (IIOP). However, the current

release of Odyssey does not add new or distinctive features from its Telescript predecessor, and the MA model is not yet stable.

2.9.5 Concordia

Concordia [31] is another MA framework built on Java. In Concordia an agent is regarded as a collection of Java objects. A Concordia agent is modeled as a Java program that uses services provided by a collection of server components, which would take care of mobility, persistence, security, communication, administration, and resources. These server components would communicate among themselves and can run in one or several Java virtual machines; the collection of these components forms the AEE at a given network node. Once arriving at a node, the Concordia agent accesses regular services available to all Java-based programs such as database access, file system, and graphics, as in Aglet. Like ARA, Concordia specifies a service bridge to provide access to legacy services. A Concordia agent is considered to have internal states as well as external task states.² The internal states are values of the objects' variables, and the external task states are the states of an itinerary object that would be kept external to the agent's code. This itinerary object encapsulates the destination addresses of each Concordia agent and the method that each would have to execute when arriving there. The designers of Concordia claim that this approach allows greater flexibility by offering multiple points of entry to agent execution, as compared to always executing an "after-move" method as in Agent Tcl, Aglet, or ARA. This concept of an externally located itinerary is similarly supported in Odyssey via Odyssey's task object. However, the infrastructure for management of these itinerary objects was not clear from the publicly available literature on Concordia.

2.9.6 Mole

In Mole [32], the agent is modeled as a cluster of Java objects, a closure without external references except with the host system. The agent is thus a transitive closure over all the objects to which the main agent object contains a reference. This island concept was chosen by the designers of Mole to allow simple transfer of agent without worrying about dangling references. Each Mole agent has a unique name provided by the agent system, which is used to identify the agent. Also, a Mole agent can only communicate with other agents via defined communication mechanisms, which offer the ability to use different agent programming languages to convert the information transparently when needed. A Mole agent can only exist in a host environment call location that serves as the intermediate layer between the agent and the OS. Mole also supports the concept of abstract location to represent the collection of distributed physical machines. One machine can contain several locations, and locations may be moved among machines. Mole limits the abstract location to denote a configuration that would minimize cost due to communication. Thus, a collection of machines in a subnet is an acceptable abstract location, whereas a collection of machines that spans cities is not. As in ARA, Mole directly proposed the concept of a system agent (server agent in ARA), which has full access to the host facilities. It is through interacting with these system agents that a given Mole agent (mobile) achieves tasks. A Mole MA can only communicate with other agents (systems and MAs) and has no direct access to resources. The uniqueness of the Mole agent model is its requirement for closure of objects, whereas other facilities such as static agent and communication are similar conceptually to other systems. What is unclear is how the Mole system enforces the closure requirement, and whether there are mechanisms to handle closure

management automatically. The concept of closure is technically convenient, but without helping tools it can be error-prone and limiting.

2.10 Distributed Mutual Exclusion

Mutual exclusion is one of the most fundamental problems in computing systems. The basic requirements for mutual exclusion were specified by Dijkstra [19] in 1965. It states, that only a single process is allowed to enter critical section at any time. If the processes are not residing on one machine and exist in a distributed environment then this problem becomes a distributed mutual exclusion problem. Following are the necessary and sufficient conditions for mutual exclusion.

1. Mutual Exclusion: Only one process can execute critical section at any time. This condition can be generalized by allowing k number of processes to execute critical section at the same time where k is an integer and $k < n$ (n = total number of competing processes). In this case it is called as k -mutual exclusion.

2. Liveliness: Any process that needs to execute critical section can do so in finite time. This condition is known as liveliness or non-starvation.

3. Fairness: All processes competing for critical section have the same probability to execute it. This condition is called as fairness.

4. Fault Tolerance: The behavior or history of any competing process outside the critical section should not effect or hinder other processes to execute critical section. This minimal fault tolerance must be an integral part of any mutual exclusion solution.

2.11 Survey of Solutions to Distributed Mutual Exclusion

Many algorithms have been presented to achieve mutual exclusion in distributed systems. These algorithms solely base on message passing. Researchers in this area have classified these algorithms based upon different criteria. Mukesh Singhal [7] has divided distributed mutual exclusion algorithms into following three classes.

Token Based: In token based algorithms a unique token is shared among the nodes. A node can enter the critical section if it possesses the token. Therefore the mutual exclusion is trivially guaranteed. A token could be a simple message, a complex data structure or an object. These protocols aim at minimizing the number of messages to locate the token. Fault models in token-based algorithms usually involve the detection and regeneration of lost token. Examples of token-based algorithms are Raymond protocol [20] and Chang-Singhal-Liu protocol [21].

Non-Token Based: These algorithms require one or more successive rounds of message exchanges among the nodes to obtain the permission to execute critical section. These algorithms are also called as permission-based protocols. This group can be further divided into two groups. The first group uses quorums or selected group of processes from which permission must be obtained before entering the critical section. The second group requires permission from all other competing processes. Examples of non-token-based algorithms are Singhal protocol [22] and Rictal-Agarawal protocol [23].

Hybrid: Theses algorithms combine the techniques of token-based and non-token-based algorithms. Hybrid algorithms are capable of combining the advantages of two mutual exclusion algorithms and offer potential of providing improved performance. Chang *et al* [25]

presented a hybrid algorithm that combines Maekawa's algorithm [24] and Singhal's dynamic algorithm [22].

Mutual exclusion algorithm in distributed systems may also be classified as centralized and distributed. In a centralized algorithm one node is responsible to serve mutual exclusion requests from all other nodes in the system. This approach suffers from many shortcomings. First of all the central node is vulnerable to failures and disconnection. Second, since at least two messages are required to grant mutual exclusion to any node, the synchronization delay could be as large as $2T$ where T is the average delay of passing a message between two nodes. This approach also suffers from locality of reference problem and central node becomes a bottleneck under heavy loads.

The performance of mutual exclusion algorithm generally measured by the following metrics.

- **Message Complexity:** Number of messages required to grant mutual exclusion
- **Response Time:** Time interval, a request waits to execute critical section after its request message to get mutual exclusion is sent out.

Performance of mutual exclusion algorithms also depends upon the load conditions. In heavy load conditions there is always a pending request to get mutual exclusion in the system. Where as in low load conditions there is seldom more than one simultaneous request for mutual exclusion in the system. Same algorithm may behave in a different manner under various load conditions. Generally performance metrics are more significant under average load conditions.

2.12 Distributed Mutual Exclusion in Mobile Agent Environment

In this section we look at distributed mutual exclusion in a mobile agent environment. As discussed Chapter-1, an application may be made up of several mobile agents, which cooperate to perform a task. During this interaction among mobile agents, they might get into a scenario where mutual exclusion is required. It is certainly a case of critical section in the mobile agent code and only one mobile agent could get into it. Some specific scenarios are discussed in the latter sections. This problem is different from that of distributed mutual exclusion in a traditional distributed environment. Therefore traditional solutions may not be applied directly to solve this problem. Following are the properties of mobile agents that bring new dimensions to this classical problem.

1. **Agent Movement:** Mobile agents are dynamic entities. They can move from one node to another to accomplish their task, whereas traditional distributed systems have static processes. To make an assumption that restricts or reduces the agility of mobile agents is simply counter to the basic idea of this paradigm. Therefore it is not desirable to fix the problem through over synchronization of agent activities.
2. **Openness of Network:** Mobile agents are gaining popularity for their ability to leverage the openness of internet or intranets. Traditional distributed systems are usually closed networks of some fixed size. Most of the existing protocols make assumptions about the size or topology and connectivity of the network. These assumptions reduce the scope of mobile agent paradigm and therefore are invalid.

3. **Multiplicity of Agents:** In a mobile agent system, agents perform transactions by moving towards the source and execute locally to leverage the benefits of local access. Agents may create new agents and new agents may move to other nodes. As a result transactions can be spread across multiple hosts without informing the node that initiated the transaction. Therefore the number and location of mobile agents participating in an application is not known. Existing protocols assume one static process on each node in the mutual exclusion competition.

As we discussed in Chapter-1, much work has been done in the design and implementation of execution environments for mobile agents, but very few of them provide high level services for coordination. In the literature survey for this dissertation, we find few papers addressing the problem of distributed mutual exclusion among cooperating mobile agents, but their approach and scope of work is different. This related work is summarized in the following paragraphs.

One work in this area is by Vera Nagamuta and Markus Endler [10]. They propose a coordination mechanism for cooperating groups of mobile agents and demonstrate its use to achieve mutual exclusion. Their mechanism is based on message diffusion or broadcast. Moreover they introduced a concept of fixed domains for groups of cooperating agents and restricted the inter domain migration of agents. This mechanism imposes an overhead of groups and a fixed proxy for a group of agents.

S. Mishra and P. Xie [6] designed and implemented synchronization mechanism for DaAgent System. They use a centralized approach to provide location independent coordination and

synchronization. A process, called “Clearing House” is chosen to be the coordination place. This solution is specifically designed for DaAgents and so is implementation dependent. It uses simple and easy to implement home-based approach. This is a centralized approach and, under heavy loads, “Clearing House “ is likely to become a synchronization bottleneck. In general, home-based schemes suffer from problems of scalability and locality.

Jiannong Cao, Xianbing Wang, and Jie Wu [8] presented a fully distributed algorithm using mobile agents to achieve mutual exclusion in a networking environment. This algorithm is designed within a specific framework for mobile agent enabled distributed server groups. This work is about solving the old problem of distributed mutual exclusion, among nodes or servers, using mobile agents. It does not address the problem of mutual exclusion among the participating mobile agents

In our research, we focus to distinguish this problem from traditional distributed mutual exclusion and propose an uncoupled and generalized solution.

Chapter 3

SOLUTION

In this chapter we propose a distributed, efficient, scalable and fault tolerant solution to distributed mutual exclusion problem in a mobile agent system. First of all we present an overview of the proposed solution. It is followed by assumptions, explanation and examples. Then we discuss correctness and analyze its efficiency. In the end we look at the unique properties of this solution.

3.1 Overview

The proposed solution is agent-based, fault tolerant, decentralized and fully distributed. The solution uses the “Arrow Distributed Directory Protocol” [1] to achieve the objective of mutual exclusion. Arrow Distributed Directory Protocol provides a scalable solution to keep track of mobile objects, as well as the ability to ensure mutual exclusion in the presence of concurrent requests. Its designers define the protocol as follows.

“The directory tree is initialized so that following the links from any node leads to the node where the object resides. When a node v wants to acquire exclusive access to the object, it sends a $\text{find}(v)$ message to $u_i = \text{link}(v)$ and sets $\text{link}(v)$ to v . When node u_i receives a find message from node u_{i+1} , where $u_{i+1} = \text{link}(u_i)$, it immediately “flips” link (u_i) to u_{i+1} . If $u_{i+1} \neq u_i$, then u_i forwards the message to u_{i+1} . Otherwise, u_i buffers the request until it is ready to release the object to v , with out further interaction with the directory.”

We have transformed this protocol to a fully agent-based and fault tolerant solution to distributed mutual exclusion problem in a mobile agent system. The basic element of our proposed solution is an agent. There are five types of agents in our model of mobile agent system: worker agents, locator agents, guide agents, mutual exclusion agent and update agents. Following section contains short description and responsibilities of each agent type. The responsibilities of these agents are also provided in Table 3.1.

Worker Agent: Worker agents are the main mobile agents in the system that perform tasks and require executing critical section.

Guide Agent: Guide agents are stationary agents. They are responsible to provide navigational information for mutual exclusion agent. They also handle the requests of worker agents to execute their critical section. Since guide agent is a stationary agent, it is also possible to replace it with an object. Most agent platforms provide communication mechanism between agents. A design decision is made to have a stationary agent to keep our solution general and suitable for agent platforms.

Locator Agent: These, mobile agents, are created by worker agents and their main responsibility is to search for the mutual exclusion agent. They perform this task with the collaboration of guide agents.

Mutual Exclusion Agent: Mutual exclusion agent is also a mobile agent and it is responsible to grant permission to the worker agents to execute the critical section.

Update Agent: These mobile agents are responsible to update redundant data structures and perform messaging to handle fault scenarios. They help system to recover from different types of failures.

There could be one and only one mutual exclusion agent in the system at any time. When a worker agent needs to execute its critical section, it must get hold of the mutual exclusion agent. To achieve this purpose the worker agent creates a locator agent. Now it is the responsibility of the locator agent to find the mutual exclusion agent and forward the request of its master worker agent. There are guide agents to help the locator agent in its quest for the mutual exclusion agent. Mutual exclusion agent grants the permission to the requesting worker agent on its turn.

Worker agents do not have any prior knowledge of other worker agents or the location of the mutual exclusion agent. The worker agents, on need basis, create guide agents during their voyage. The details of this mechanism are explained later in this chapter.

There are three main requirements of any mutual exclusion solutions: exclusion, fairness and liveness. The underlying protocol, Arrow Directory Protocol, in our proposed solution guarantees all these requirements are fulfilled. We discuss the correctness of this protocol later in this chapter.

The elements and their tasks that are part of this distributed mutual exclusion solution in a mobile agent system are described in the following table.

Table 3.1 – System Agents and Responsibilities

ELEMENT NAME	DESCRIPTION
Worker Agent	<ul style="list-style-type: none"> • Main agents in the system, which perform tasks. • Only agents in the system with critical sections and may request for mutual exclusion. • Need permission from mutual exclusion agent to execute critical section. • Unaware of the existence, location and state of other worker agents. • Unaware of the location and state of mutual exclusion agent.
Locator agent	<ul style="list-style-type: none"> • Created by worker agents when they need to execute their critical section. • Travel from node to node with the navigational help of guide agents to look for mutual exclusion agent. • The only responsibility is to find the location of mutual exclusion agent and forward the request for mutual exclusion on behalf of its creator worker

	<p>agent.</p> <ul style="list-style-type: none"> • Disposed off as soon as successfully handover the request for mutual exclusion.
Guide Agent	<ul style="list-style-type: none"> • Created by worker agents on their voyage. • Hold the navigational information for mutual exclusion agent. • Communicate with locator agents to help them find mutual exclusion agent. • Store requests from local worker agents for mutual exclusion and forward them to mutual exclusion agent on its arrival.
Mutual Exclusion Agent	<ul style="list-style-type: none"> • Created at the initiation of system. Only one mutual exclusion agent in the entire system at any time. • Responsible for granting permission to the worker agents to execute their critical section. • Moves to a remote node if the mutual exclusion request is from a remote worker agent. • On arrival at a new node, communicates with the

	guide agent to get information of pending mutual exclusion request.
Update Agents	<ul style="list-style-type: none"> • Created by guide agents to handle different fault scenarios. • Main objective is to update data structures and perform messaging in case of faults. • Detailed responsibilities are discussed in the fault tolerance related section of this chapter.

3.2 Assumptions

In this section we present the assumptions about the system and agents necessary to make our solution work properly. These assumptions are helpful to evaluate the limits and properties of the solution. The driving force in our solution is not to make any assumption that reduces or restricts the flexibility provided by this paradigm. Since our effort is to make a generic solution that could be implemented in any mobile agent platform, assumptions about the services provided by the system are very limited. First we list the general assumptions and then we present the fault related assumptions.

3.2.1 General Assumptions

1. It is assumed that agents are free to move on the network at will. Any agent requesting for mutual exclusion or executing the critical section puts no restriction on the movement of other agents.
2. Direct communication between agents is possible on any node. It means that physically present on the same node, two agents can send and receive messages from each other.
3. Agents are able to communicate and get services from the agent server or from the host environment. These services are related to locate an agent, by name or by some unique property, present on the same node.

4. Agent server provides an event model to monitor the flow of agents. Events like arrival, departure and demise of agents are detectable and actions can be performed on them.

3.2.2 Fault Assumptions

The proposed solution is independent of network organization. In the following assumptions, word adjacent is logical and refers to the adjacency in the underlying directory structure.

1. All failures are permanent.
2. Links are reliable and a failed delivery of an agent to any node is due to the node crash.
3. Worker agents and locator agents are robust and do not fail.
4. Two adjacent nodes or guide agents on the adjacent nodes do not crash simultaneously.
5. If node-A and node-B are to be granted mutual exclusion one after the other, we call them *ME-successive-nodes*. *ME-successive-nodes* or guide agents on *ME-successive-nodes* do not fail simultaneously.

Assumption-4 and assumption-5 limit the redundancy of the data structures and computation that provide fault tolerance. We may enlarge the size and depth of redundant data structures and relax these assumptions.

3.3 Description of the Solution

Figure-3.1 depicts the diagrammatic representation of different kind of agents in the system.

We will use this notation in this notation in the other diagrams of this thesis.

Figure 3.1 - Symbols for Agent Representation

Now we describe the details of our solution. First we present the solution in the absence of any fault and later we explain different fault scenarios. The whole process of providing solution to mutual exclusion problem can be divided into following four stages.

1. Initialization
2. Request
3. Grant
4. Release

3.3.1 Initialization

On any node where a mobile agent based application is initiated and worker agents are created, two other agents mutual exclusion agent and guide agent are also created. When a worker agent moves to a new node, it creates a guide agent on its arrival, if no guide agent is already

present at that site. In this way directory structure is initialized in a lazy manner and as many guide agents are created as needed.

3.3.2 Request

mutual exclusion agent, locator agent updates pointers of guide agents to point to new direction.

Figure-3.3 shows that worker agent at N_8 needs to execute its critical section and creates a locator agent. The locator agent traces the path, provided by the guide agents, to find mutual exclusion agent at N_5 . Locator agent shown with broken lines shows the presence in past.

Figure 3.3 – Request for Mutual Exclusion

3.3.3 Grant

When a mutual exclusion agent is free, it informs the guide agent about its availability. If there exists a queued request from any worker agent, mutual exclusion agent is moved to the requesting node. If there is no pending request for mutual exclusion, mutual exclusion agent stays at the same node.

Figure-3.4 shows that mutual exclusion agent has migrated to N_8 from N_5 to grant mutual exclusion on that node.

Figure 3.4 – Granting mutual Exclusion

3.3.4 Release

When a worker agent, waiting to execute its critical section, gets hold of mutual exclusion agent, it completes its task and releases the mutual exclusion agent to guide agents. If the guide agent has any request in its queue, mutual exclusion agent moves to requesting agent. Otherwise it stays there until it gets new request.

3.4 Examples of Solution

Here we present some examples to illustrate our solution. First example is simple one to describe the basic functionality of our Arrow Directory Protocol based solution. Second example depicts a concurrent scenario of multiple worker agents trying to execute their critical section at the same time.

3.4.1 Simple Example

Figure 3.5 - Simple Example (1)

Figure-3.5 shows a scenario with mutual exclusion agent at N_1 and worker agents at N_1 , N_2 , N_5 , N_7 , N_8 , N_9 , N_{11} and N_{13} . Guide agents are not shown, instead arrow direction shows the navigational information provided by them. If the worker agent at N_{11} needs to execute its critical section, it creates a locator agent. The locator agent leaves to search for mutual exclusion agent with the navigational help of guide agents.

Figure-3.6 shows the state when locator agent just arrives at N_3 . New request for mutual exclusion arriving at N_3 is directed towards N_4 and the one arriving at N_2 is directed towards N_1 . The request arriving at N_2 first, has a preference to possess mutual exclusion agent. This scenario is explained in the next section, when we describe the concurrent behavior of this protocol.

Figure 3.6 - Simple Example (2)

Figure-3.7 shows the arrival of the locator agent at N_1 but the mutual exclusion agent is not yet released to serve worker agent at N_{11} . Any new request for mutual exclusion is directed towards N_{11} .

Figure 3.7 - Simple Example (3)

When the mutual exclusion agent at N_1 finishes its job, it moves itself directly to N_{11} . The movement of the mutual exclusion agent to N_{11} does not follow the directory path. Figure-3.8 shows the final state of this example.

Figure 3.8 - Simple Example (4)

3.4.2 Concurrent Example

This example illustrates a scenario where multiple agents try to acquire mutual exclusion agent. Consider the previous simple example (Figure 3.6) when the locator agent from N_{11} arrives at N_3 , worker agent at N_8 needs to execute its critical section and dispatches its locator agent. This request is blocked with the guide agent at N_{11} and all intermediate guide agents update their navigational information as depicted in the following Figure-3.9. Any new request for mutual exclusion arriving at N_3 is directed towards N_8 and the one arriving at N_2 is directed towards N_1 . The request arriving at N_2 first, has a preference to possess mutual exclusion agent.

Figure 3.9 - Concurrent Example (1)

Figure 3.10 shows a more complex scenario when agent at N_7 needs to execute its critical section and its locator agent arrives at N_2 before that of N_{11} . This request is served first.

Figure 3.10 - Concurrent Example (2)

Figure 3.11 - Concurrent Example (3)

Figure 3.12 - Concurrent Example (4)

Figure 3.11 shows the scenario when the locator agent of N_{11} arrives at N_7 and its request is blocked there. Figure 3.12 shows another scenario when agent at N_5 needs to execute its critical section and its locator agent arrives at N_8 .

3.5 Fault Tolerance

A system is considered fault tolerant if the behavior of the system, despite the failure of some its components, is consistent with its specifications. Fault tolerance uses *protective redundancy* to mask failures. That is, the system contains components that are not needed if no fault tolerance is to be supported. These redundant components are used to avert system in case some components fail [36].

In a distributed mobile agent environment a variety of faults may occur. Detection and recovery from faults impose overhead on the solution. Faults could be network related like link failures, crashed sites related like node failures and agent environment related like agent failures. The number of possible faults or points of failure, in a distributed environment, is very large and it is close to impossible to design a fully fault-tolerant system. The fault model considered for our proposed solution is limited by the assumptions described in Section-3.2. Update agents, introduced in Section-3.1, are used to accomplish the task of fault tolerance. These agents update different data structures deliver messages. These tasks are simple and straightforward. To limit the scope of this work, a detailed design and responsibility allocation of different update agents is not considered.

In this section, we propose a set of rules that provides a mechanism to detect and recover from the faults. Then we present, in detail, the possible scenarios of our fault model and explain, how the presented rules are applied to recover from the faults. This section also contains few definitions that make it more convenient to explain the rules and their application.

3.5.1 Definitions

Following are the definitions that we use to make our discussion, about the rules and their application, more clear and comprehensible.

ME (Mutual Exclusion) Adjacent Node: If a guide agent at node A points towards node B for mutual exclusion agent, node A is the *ME-adjacent-node* of node B.

Primary ME (Mutual Exclusion) Direction: If a guide agent at node A points towards node B for the mutual exclusion agent, the pointer of guide agent at A is called as *Primary-ME-Direction*.

Secondary ME (Mutual Exclusion) Direction: If a guide agent at node A points towards node B for the mutual exclusion agent, the *Primary-ME-Direction* of guide agent at B is called as the *Secondary-ME-Direction* of node A.

3.5.1 Rules for Fault Tolerance

This section contains rules to handle fault conditions. These rules explain redundant data structures and computation required to detect and recover from the fault conditions.

Rule1: Each guide agent maintains a complete list of its *ME-adjacent-nodes*. Figure-3.13 illustrates this rule. The *ME-adjacent-nodes* of N_1 are N_3 , N_5 and N_6 . The *ME-adjacent-nodes* of N_3 are N_2 and N_4 .

Figure 3.13 - Rule-1

Rule-2: Each guide agent keeps information of its *Secondary-ME-Direction* along with *primary-ME-Direction*. Figure-3.14 illustrates this rule. A full line shows *primary-ME-Direction* and a dashed line depicts *Secondary-ME-Direction*. *Primary-ME-Direction* of N_1 is N_2 and that of N_2 is N_3 . *Secondary-ME-Direction* of N_1 is N_3 and that of N_2 is N_4 .

Figure 3.14 – Rule-2

Rule-3: If communication through *primary-ME-Direction* fails at any node, *secondary-ME-Direction* becomes *primary-ME-Direction* and all of its *ME-adjacent-nodes* are notified to update their *secondary-ME-Direction*. The *secondary-ME-Direction* of guide agent at that node is also updated.

Rule-4: If a request for mutual exclusion is blocked with a guide agent, this blocked request is immediately registered with the guide agent of first node in its *ME-adjacent-nodes*.

Rule-5: The arrival or departure of mutual exclusion agent at any node is immediately notified to the guide agent of the first node in its *ME-adjacent-nodes*. The guide agent, which receives arrival notification, periodically monitors the state of the notification sender. The guide agent, which receives departure notification, stops monitoring the state of the notification sender.

Rule-6: On arrival of the mutual exclusion agent at any node, list of *ME-adjacent-nodes* is dispatched to the guide agents of *ME-adjacent-nodes*.

Rule-7: Locator agent or mutual exclusion agent returns to its sender node, if it detects failure of any component on arrival to a new node.

3.5.2 Fault Scenarios

This section describes the possible fault scenarios of our model. We describe how the rules described in the previous section provide fault tolerance. Following are the three major possibilities of failure.

- Case A: Node Failure
- Case B: Guide Agent Failure.
- Case C: Mutual Exclusion Agent Failure

Case A: Node Failure

In our fault model, there could be four different scenarios in case of a node failure. These sub-cases are discussed in the following paragraphs

Case A-1: A node (in possession of a guide agent, without any blocked request) fails

In this case the navigational information to locate mutual exclusion agent is lost and all *ME-adjacent-nodes* need an alternative to locate mutual exclusion agent. This case is handled by Rule-2 and Rule-3. The guide agents at *ME-adjacent-nodes* of this crashed node use their *Secondary-ME-Direction* to locate mutual exclusion agent.

Case A-2: A node (in possession of a guide agent, with blocked request) fails

In this scenario, not only the navigational information is lost, but also the information about the blocked request is missing. This case is handled by Rule-2, Rule-3 and Rule-4. The recovery process of navigation information to locate the mutual exclusion agent is the same as described in Case A-1. If the mutual exclusion agent is unable to go the node with a blocked request, it is dispatched to the first member of its *ME-adjacent-nodes*. This node (Rule-4) has a record for the lost blocked request and dispatches the mutual exclusion agent to serve the request.

Case A-3: A node (in possession of mutual exclusion agent, without any blocked request) fails

According to Rule-5, first member of *ME-adjacent-nodes* monitors the state of the node that possesses the mutual exclusion agent. In case of failure detection, it generates another mutual exclusion agent and notifies other members of the *ME-adjacent-nodes*.

Case A-4: A node (in possession of mutual exclusion agent, with blocked request) fails

The mutual exclusion agent is generated the same way as described in Case A-3. Since the first node in *ME-adjacent-nodes* has a record of the blocked request (Rule-4), it dispatches the mutual exclusion agent to serve it.

Case B: Guide Agent Failure

Guide agents provide contact point to the mutual exclusion agent, maintain navigation information for mutual exclusion agent and keep record of any blocked request. Therefore, the crash of a guide agent results in the loss of very vital system information, and it requires a mechanism to rebuild or to retrieve the lost information. In the presented solution, a guide agent may be in one of the following three states after its creation. The details of failure scenario in each state and its recovery procedure are described below.

Case B-1: Guide agent (without any blocked request) fails

In this case, locator agent returns to its sender (Rule-7) and uses the secondary link as in Case A-1.

Case B-1: Guide agent (with blocked request) fails

In this scenario, mutual exclusion agent returns to its sender (Rule-7) and uses the technique of Case A-2.

Case C: Mutual Exclusion Agent Failure

Guide agent at the same node monitors the status of the mutual exclusion agent. On detection of the mutual exclusion agent failure, it generates a new one.

3.6 Mutual Exclusion Solution Pseudo Code

This section presents the pseudo code for different elements involved in our proposed solution. This code provides the responsibilities, behavior and response to different messages for each agent in the system. The code does not include the fault conditions we discussed in the previous section.

3.6.1 Worker Agent Pseudo Code

Worker agents are application agents who perform different tasks. They have critical sections and need mutual exclusion to execute them. In this section, we present the partial pseudo code for worker agent that is required to execute its critical section.

```
States S = {Created, Arrived, Processing, Waiting}
Sinit = {Created, Arrived}

Created
    On_Creation
    Begin
        CurrentLocation = Host Name
        PreviousLocation = Null
        Guide_Agent_Proxy = Get Guide Agent Proxy From
            The Environment
        If (Guide_Agent_Proxy == NULL)
            // There is no Guide agent already present
            Create_Guide_Agent (CurrentLocation)
        Endif
    End
Processing
Processing
    Begin
        Do Some Task      // Application specific task

        // Beginning of Critical Section
        Ceate_Locator_Agent (CurrentLocation)
    End
Waiting
Waiting
    Begin
        // Got permission to enter into Critical Section
        Received (GRANT)
```

```

        Do Some Task      // Application specific task
        //End of Critical Section
        Send(RELEASE) to Mutual_Exclusion_Agent
        Do Some Task      // Application specific task
    End
Processing
Processing
    On_Dispatch (DestinationURL) // Agent is moving to new location
    Begin
        Previous Location = Current Location
        Current Location = DestinationURL
        Dispatch(DestinationURL)
    End
Arrived
Arrived
    On_Arrival      // Agents has arrived on a new location
    Begin
        Guide_Agent_Proxy = Get Guide_Agent_Proxy Form
                           the Environment
        If(Guide_Agent_Proxy == NULL)
            // There is no Guide agent Already present
            Create_Guide_Agent (CurrentLocation)
        Endif
    End
Processing

```

3.6.2 Locator Agent Pseudo-Code

States S = {Created, Arrived, Processing, Waiting, Disposed}
 S_{init} = {Created, Arrived}

```

Created
    On_Creation (CurrentLocation)
    Begin
        PrimaryDirection = CurrentLocation
        RequestingURL = CurrentLocation
        Send (REQUEST, PrimaryDirection, RequestingURL)
        to Guide Agent
    End
Waiting
Waiting
    //Mutual Exclusion Agent is not found
    Receiving(MOVE, PrimaryDirectionMEA) from Guide Agent
    Begin
        PrimaryDirection = Host_Name
        Dispatch(PrimaryDirectionMEA)
    end
Arrived
Waiting
    // Mutual Exclusion Agent is found

```

```

        Receiving (DISPOSE) from Mutual Exclusion Agent
        Begin
            Dispose
        End
Disposed
Arrived
    On_Arrival        // Arriving at a new location
    Begin
        Send(REQUEST,PrimaryDirection,RequestingURL)
            to Guide Agent
    End
Waiting

```

3.6.3 Guide Agent Pseudo Code

```

States S = {Created, Processing}
Sinit = {Created}

Created
    On_Creation(CurrentLocation)
    Begin
        PrimaryDirectionMEA = CurrentLocation
        PossessMEA = false
        ContainerOfLocalRequests = NULL
        NextURL = NULL
    End
Processing
Processing
    Begin
        // Received request from Locator Agent
        Receiving(REQUEST,PrimaryDirection,RequestingURL)
        ProcessRequest(REQUEST,PrimaryDirection,RequestingURL)
    End
Processing
Processing
    Begin
        // Mutual Exclusion Agent has arrived
        Receiving (ARRIVED) from Mutual Exclusion Agent
        PossessMEA = true
        Send(REQUEST,ContainerOfLocalRequests,NextURL)
            to Mutual Exclusion Agent
        ContainerOfLocalRequests = NULL
        NextURL = NULL
    End
Processing
Processing
    Begin
        // Mutual Exclusion Agent is departing
        Receiving (DEPARTING) from Mutual Exclusion Agent
        PossessMEA = false
    End

```



```

        End
Processing

Procedure ProcessRequest (REQUEST, PrimaryDirection, RequestingURL)
    Begin
        If (PrimaryDirectionMEA == Host_Name)
            If (PossessMEA == true)
                Send (REQUEST, RequestingURL)
                    to Mutual_Exclusion_Agent

            Else
                If (REQUEST From_Remote_WorkerAgent)
                    NextURL = RequestingURL
                    PrimaryDirectionMEA =
                        PrimaryDirection

                Else
                    Add REQUEST to
                        ContainerOfLocalRequests

                Endif
            Endif
        Else
            Send (MOVE, PrimaryDirectionMEA) to
                Locator Agent
            PrimaryDirectionMEA = PrimaryDirection
        Endif
    End
End

```

3.6.4 Mutual Exclusion Agent Pseudo Code

States S = {Created, Arrived, Processing, Waiting}
 S_{init} = {Created, Arrived}

```

Created
    On_Creation
    Begin
        busy = false
        ContainerOfPendingLocalRequests = NULL
        NextCandidateURL = NULL
    End
Waiting
Arrived
    On_Arrival
    Begin
        Send (ARRIVED) to Guide Agent
        Busy = true
        Send (GRANT) to WorkerAgent
    End
Waiting
Waiting

```

```

    Receiving (REQUEST, ContainerOfLocalRequests, NextURL)
        From Guide Agent
    Begin
        ContainerOfPendingLocalRequests = ContainerOfLocalRequests
        nextCandidateURL = NextURL
        Process_REQUEST
    End
Waiting
Waiting
    Receiving (REQUEST, RequestingURL)
    Begin
        If (REQUEST From_Remote_Agent)
            NextURL = RequestingURL
        Else
            Add_REQUEST to ContainerOfPendingLocalRequests
        Endif
        Process_REQUEST
    End
Waiting/Arrived
Waiting
    Receiving (RELEASE)
    Begin
        Busy = true
        Process_REQUEST
    End
Waiting/Arrived
Procedure Process_REQUEST
    Begin
        If (busy == false AND ContainerOfPendingLocalRequests
            Is Not_Empty)
            Busy = true
            Get Next Worker Agent from
                ContainerOfPendingLocalRequests
            Send (GRANT) to the selected Worker Agent
            Become Waiting
        Else
            If (NextCandidateURL != NULL)
                Dispatch (NextCandidateURL)
                Become Arrived
            Endif
        Endif
    End

```

3.7 Correctness

The solution proposed in previous sections is based upon the Distributed Arrow Directory Protocol. A formal proof of correctness for this protocol can be found in Section-4 of paper presented by Demmer and Herlihy [1]. We have transformed this protocol to suit an agent

environment. In this section we prove that our presented solution is correct and fulfills the requirements of mutual exclusion, fairness, liveness and minimal fault tolerance, as described in Section-2.10.

An agent can enter into the critical section only if it possesses the mutual exclusion agent. We need to prove that only one agent could possess mutual exclusion agent at any time. Following three facts prove the exclusive ownership of mutual exclusion agent

1. There is only one mutual exclusion agent in the system any time.
2. Mutual exclusion agent grants ownership when it is not occupied.
3. As soon as mutual exclusion agent grants ownership to any worker agent, it becomes occupied and these actions are atomic.

If the mutual exclusion agent resides on the same node with the requesting locator agent and no external request is already blocked, request is queued. Mutual exclusion agent serves all requests in the queue before it moves to serve any external request. Distributed Arrow Directory Protocol guarantees that each locator agent finds the mutual exclusion agent or blocks itself with a guide agent that expects the arrival of mutual exclusion agent. This mechanism ensures liveness of the presented solution.

Mutual exclusion agent serves the requests on first come first serve basis. No preemption is allowed of the requests blocked at mutual exclusion agent or at any guide agent. Similarly order of local requests collected in a queue cannot be changed. This strategy proves the fairness of presented solution. However this fairness is relative and depends on the time, when the

request is received by the mutual exclusion agent or guide agent and not on the time when it was generated. The delay between request generation and its grant depends upon the network congestion.

Since this solution is based upon the autonomous agents, the behavior of any worker agent outside the critical section does not affect the ability of other agents to execute their critical sections.

3.8 Cost Analysis

Performance of our proposed solution is measured in terms of agent migrations. It is possible to calculate theoretical values for the number of agent migrations, assuming a sequential execution. Three agents participate to provide mutual exclusion for any application; locator agent, guide agent and mutual exclusion agent. Guide agents are stationary and they stay on the same server where they were created. Mutual exclusion agent migrates once per critical section invocation from a remote agent. The movement of mutual exclusion agent is direct, through the network, to the requesting server. The only agent migration affecting the performance is the migration of locator agent.

Number of Migrations of locator agent depends upon the stretch of worker agents. This stretch can be measured as depth of minimum spanning tree (MST) of worker agents. The worst case of this MST is a straight line (MST for a ring). The lower the worker agents MST stretch, the lesser is the number of locator agent migrations. If m is the number of worker agents seeking mutual exclusion and n is the depth of MST for these worker agents, the complexity of proposed solution is bounded by $O(mn)$.

In reality, agent migrations are not sequential. Locator agent movements for different worker agents occur in parallel and exploit the locality of reference. This means that time required for mutual exclusion solution does not depend upon the number of agent migrations, since many occur at the same time. Execution time estimates depend upon many other factors such as network congestion, processing speed of servers, efficiency of agent environment and mechanism of local communication among agents.

3.9 Solution Properties

The presented solution is designed to meet the challenges of mobile agent environment. In this section we identify the properties of this solution. These properties help to understand and appreciate utility and benefits of the solution.

3.9.1 Distributed Solution

A common and straightforward approach to this problem of mutual exclusion is to make a centralized mutual exclusion server. That server maintains a large data structure, collects all the requests in it and serves them one by one. This approach has problems of scalability and locality of reference, among many others. Contrary to this approach our proposed solution is fully distributed. We do not collect request on one server, rather each request for mutual exclusion stored on different server.

3.9.2 Decoupling

This solution provides full spatial decoupling among participating mobile agents. Worker agents do not need to know the name, number or location of one another. Worker agent just

needs to create a locator agent and then it becomes the responsibility of proposed mechanism to fulfill the mutual exclusion request.

3.9.3 Network Organization Independence

The proposed solution is network organization independent and makes no assumption about the topology or size of the network. Therefore the solution is scalable for any kind and size of network.

3.9.4 Fault Tolerance

The presented solution is fault tolerant and provides recovery mechanism for crashed nodes and agents. The fault tolerance imposes an overhead on the solution. We consider a limited fault model.

3.9.5 Protocol Encapsulation

Mobile agents encapsulate complex protocol and computing. This provides a neat and easy to implement solution. Our solution is fully comprised of autonomous agents.

Chapter 4

PROTOTYPE IMPLEMENTATION AND TESTS

The solution presented in the previous chapter is implemented using the Aglets Software Development kit (ASDK) [4,28], an agent-programming environment developed at the IBM Tokyo Research Laboratory. In this chapter we describe the details of platform selection, design decisions and implementation issues. We also present the results of various tests, performed under different load conditions, to measure the performance of our proposed solution.

4.1 Objectives and Scope

The main objective of the prototype implementation is to demonstrate that the solution provided in the previous chapter is applicable on a contemporary agent system. A secondary goal is to perform experiments and gather data to measure the actual performance of the protocol. The results obtained are also compared with the theoretical analysis described in Section-3.7.

This implementation is not an efficient and readily usable general-purpose software tool to provide mutual exclusion in Aglet environment. This work, however, is capable of becoming a base for such a toolkit. This implementation does not cover the fault-tolerant scenarios explained in Chapter-3. There is clearly a room to improve the performance and efficiency of this prototype.

4.2 Environment Selection

There are many options to select a platform for prototype implementation. We discussed many platforms and their properties in Chapter-2. All those environments are good candidates for prototype implementation. Another option is to make an original simulation engine. Since a lot of agent environments are available and are being actively used for application development, the development of an original simulation environment is not necessary. Our proposed solution requires limited services from the environment. Therefore any platform, that meets the assumptions listed in Chapter-3, can be used for the prototype.

Java has become one of the most popular languages for mobile agents. The properties like platform independence, secure execution, multithread programming, object serialization and dynamic class loading make Java a suitable language for mobile agent environments. Since most of the contemporary agent systems are developed in Java, we prefer to select a Java based platform. IBM's ASDK [4,28], a Java based environment, meets the requirements of our solution in addition to offering many interesting properties discussed in the next section.

4.3 ASDK Overview

The Aglet framework was initially developed and maintained as Aglets Software Development Kit (ASDK) by the IBM Tokyo Research Laboratory. Many enterprises around the world have made a considerable investment in aglet technology with respect to their own application development, as well as in providing feedback to the ASDK development team. Now this toolkit has become an open source initiative. Information about further development and documentation is available at <http://aglets.sourceforge.net>.

The Aglets framework is one of the well-known distributed application frameworks. The term Aglet is derived from agent and applet. This framework is based upon a well-designed and small API. Programmers need to override predefined methods to add desired functionality. With this framework, it is possible to design truly distributed applications, that is, applications whose logic and dependencies are shared among many network hosts.

The object model of Aglet is designed to benefit from the agent characteristics of Java. Following are the important characteristics of an Aglet.

- It has its own thread of control.
- It is event-driven.
- It communicates by message passing.

The basic abstractions of Aglet API are Aglet, Proxy, Context and Identifier. Proxy is a representative of an Aglet. It protects an Aglet from direct access of its public methods. Identifier is unique for each Aglet and immutable throughout its lifetime. Context closely corresponds to home for an Aglet. It is a stationary object and provides a mean to manage and maintain running Aglets. The fundamental operations of an Aglet are creation, cloning, dispatching, retracting, activation, deactivation and disposal.

Aglets communicate by message passing. Messages can be sent among Aglets in the same place, that is, locally, or to Aglets at remote contexts. This framework also supports both synchronous and asynchronous messaging. It allows sending one way, two way or future

messages. However, for this work we rely on local, asynchronous and one way messaging only.

4.4 Implementation Details

Implementation of our solution is exactly the same as detailed in the high-level design in Chapter-3. Locator agent, guide agent and mutual exclusion agents are implemented as different classes and they interact by messaging when present at the same host. A helping class MEProperties is also implemented to contain system properties. Code listing of these classes is provided in Appendix-I. We use a package zerf.aglet [37] to log the results and necessary information of our experiments. This package provides a stationary aglet called as StationaryLog. Its purpose is to log messages (to a file) sent by other agents that were instantiated in the same context as StationaryLog, but are roaming the Internet.

We also implement a worker agent to test the prototype. The worker agent has a vector of different hosts as its data member and visits these hosts with a random order. It executes its critical section at each host, it visits, and the mutual exclusion is guaranteed by our presented solution.

4.5 Test Methodology

Tests are performed to demonstrate the working of our proposed solution and to measure its performance. This allows us to compare the actual results with the theoretical bounds. Two metrics, number of locator agent migrations and response time, are used to measure the performance. Response time is the time interval between a worker agent requests for the

mutual exclusion and gets the permission. We also keep track of the minimum and maximum response time for each request.

Tests are performed with different combinations of three elements to obtain a range of measurements. These three elements are number of servers (hosts), number of requests for mutual exclusion and load conditions. Configuration of five and ten hosts is used with three different number of mutual exclusion requests. We consider two load conditions; heavy load and light load.

Heavy Load: There are simultaneous requests for mutual exclusion at any time.

Light Load: There are no simultaneous requests for mutual exclusion at any time.

Each test is performed under both, heavy and light, load conditions to compare the load impact. A sample of the execution trace of the experiments performed is provided in Appendix-II.

4.6 Test Results

In this section, we present the results of experiments that we performed to test the prototype. We have performed four different sets of experiments. Each set of experiments was repeated by increasing the number of critical section invocations. The results for each set are reported and illustrated graphically in a separate section.

4.6.1 Test Case 1: 10-Nodes Under Heavy Load Conditions

This section contains the results of 10-nodes test, performed under heavy load conditions.

These test results are also illustrated graphically.

No. Of ME Requests	Min. Response Time (ms)	Max. Response Time (ms)	Ave. Response Time (ms)	No. Of Agent Migrations	Ave. Agent Migrations (Agent Migrations/ ME Requests)
25	300	25576	8605	38	1.5
42	621	29452	9763	71	1.7
80	722	34200	11555	145	1.8

Table 4.1 - Response Time & Agent Migrations Under Heavy Load (10-Hosts)

Figure 4.1 - Response Time Under Heavy Load (10-Hosts)

Figure 4.2 - Average Agent Migrations Under Heavy Load (10-Hosts)

4.6.2 Test Case 2: 10-Nodes Under Light Load Conditions

This section contains the results of 10-nodes test, performed under light load conditions.

These test results are also illustrated graphically.

No. Of ME Requests	Min. Response Time (ms)	Max. Response Time (ms)	Ave. Response Time (ms)	No. Of Agent Migrations	Ave. Agent Migrations (Agent Migrations/ ME Requests)
30	110	20700	8555	58	1.9
49	100	22733	7020	65	1.3
70	130	19468	7414	88	1.3

Table 4.2 - Response Time & Agent Migrations Under Light Load (10-Hosts)

Figure 4.3 - Response Time Under Light Load (10-Hosts)

Figure 4.4 - Average Agent Migrations Under Light Load (10-Hosts)

4.6.3 Test Case 3: 5-Nodes Under Heavy Load Conditions

This section contains the results of 5-nodes test, performed under heavy load conditions.

These test results are also illustrated graphically.

No. Of ME Requests	Min. Response Time (ms)	Max. Response Time (ms)	Ave. Response Time (ms)	No. Of Agent Migrations	Ave. Agent Migrations (Agent Migrations/ ME Requests)
16	340	11150	4951	17	1.1
27	355	13309	5350	28	1.0
46	400	18100	9898	54	1.2

Table 4.3 - Response Time & Agent Migrations Under Heavy Load (5-Hosts)

Figure 4.5 - Response Time Under Heavy Load (5-Hosts)

Figure 4.6 - Average Agent Migrations Under Heavy Load (5-Hosts)

4.6.4 Test Case 4: 5-Nodes Under Light Load Conditions

This section contains the results of 5-nodes test, performed under light load conditions. These test results are also illustrated graphically.

No. Of ME Requests	Min. Response Time (ms)	Max. Response Time (ms)	Ave. Response Time (ms)	No. Of Agent Migrations	Ave. Agent Migrations (Agent Migrations/ ME Requests)
18	200	14351	4827	21	1.2
25	355	12969	3984	28	1.1
47	180	31316	6200	55	1.2

Table 4.4 - Response Time & Agent Migrations Under Light Load (5-Hosts)

Figure 4.7 - Response Time Under Light Load (5-Hosts)

Figure 4.8 - Average Agent Migrations Under Light Load (5-Hosts)

4.7 Analysis of Test Results

The experiment results presented in the previous section prove that the solution is able to meet its required properties. We can draw following important conclusions.

- The number of agent migrations per critical section invocation (average agent migrations) is very low and well beyond the theoretical bound provided in Chapter-3. This finding attributes to the efficiency of solution.
- Increase in the average agent migrations is nominal as compared to the increase in network size or load conditions. This shows that the solution is scalable and suitable for open networks.
- Average agent migrations decrease under heavy load conditions. This fact shows that solution exploits the locality of reference.
- Response time increases under heavy load conditions and with the increase in network size. This property also depends on many other factors like efficiency of links.

Chapter 5

CONCLUSIONS AND FUTURE WORK

In this chapter we conclude our work by providing a summary of the thesis contribution. We also compile the conclusions drawn from the work completed for this dissertation. Finally we look at the prospects of future work in this direction.

5.1 Thesis Summary

In this dissertation we defined the problem of distributed mutual exclusion among cooperating mobile agents, and distinguished it from the traditional distributed mutual exclusion problem in a client-server environment. We have also identified the factors that make existing solutions unsuitable for this problem of mobile agents. We have presented a background study of distributed systems, mobile agents and distributed mutual exclusion. Then we proposed a distributed, scaleable and fault tolerant solution. We provided specific examples to demonstrate the working of the proposed solution. Pseudo code for different elements of this solution has also been given to implement it quickly in any available platform. We discussed the correctness of given solution in terms of mutual exclusion, liveness, fairness and fault tolerance. Complexity analysis provided the cost of solution in terms of agent migrations. We also discussed the upper bound on number of agent migrations in the worst case. We identified the unique properties and advantages of the proposed solution. The Presented solution has been implemented on a specific agent platform, IBM Aglets. Tests have been performed to find out the actual cost of the solution and have been compared with the

theoretical bounds. We have also drawn conclusion and have provided the prospects for future work in this direction.

5.2 Conclusions

The goals and objectives for this work, specified in Chapter-1, have been achieved satisfactorily. We identified the problem of distributed mutual exclusion among cooperating mobile agents and presented a solution, described in Chapter-3. This solution leverages the advantages of mobile agent paradigm and exhibits the desired properties, outlined in Chapter-1. It is distributed (non-centralized), scaleable, fault tolerant and does not suffer from locality of reference. Since this solution does not impose any restriction about the network sizes, network organization or agent movement, it suits the agile and open nature of mobile agent environment. It is decoupled from any specific agent environment because it requires very basic services from the platform.

The prototype implementation with IBM Aglets demonstrates that the presented solution is practical. Several experiments have been performed by varying different elements of the system. Test results show that the solution cost is within the theoretical bounds. We could not find similar solutions for this problem, therefore the evaluation and comparison of our solution is inconclusive.

The fault tolerance aspect of this solution is limited. The limited scope of this work and unbounded nature of faults forced us to assume many restrictions. There is, clearly, a room for improvement and enhancement in this direction.

5.3 Future work

Research or thinking about problems and their solutions is a continuous process. A second look and more thinking bring new aspects and directions to our attention. In this section we present further directions to continue the work presented in this thesis.

The solution we presented allows only one agent to execute its critical section at a time. There could be scenarios where more than one agent may enter critical section at the same time. This problem is K-mutual exclusion (where K is an integer) problem and has been studied in traditional distributed systems. The solution we present may be generalized for K-mutual exclusion problem.

The solution proposed by this thesis is a fault tolerant solution. We investigate certain fault conditions, critical in our opinion, and address them. But we do not cover many other fault conditions to limit the scope of this work. Further work is necessary to in this direction to enhance the scope of this solution. One very important improvement could be to consider the failure of locator agent before it finds the mutual exclusion agent or blocks it self with a guide agent. In this scenario there is need to undo the changes, made by locator agent, in the directory structure.

In mobile agents another type of synchronization, in addition to execution synchronization, is identified called as location synchronization. In mobile computing environment some time there is a need to coordinate the location of execution of cooperating mobile agents. One example of location synchronization is that only k-number of mobile agents can visit one node

at the same time. Solution proposed by this thesis may be generalized to provide location synchronization.

We implement our solution only on one platform, Aglets, provide by IBM. It is interesting to implement this solution on other, especially on non-Java, platforms. Other implementations would allow comparing the behavior of the proposed solution on different agent platforms. This comparison ranges from ease of implementation to experimental results.

In our solution we have created many agents to provide mutual exclusion. Some of these agents, like the locator agents, are destroyed after completion of their task, while others, like the guide agents, stay there and need to be disposed off. There is need to look into the garbage collection issue of unnecessary agents.

APPENDIX-I

Code Listing

Mutual Exclusion Agent

```
package fayyaz.mutualexclusion;

import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import java.net.*;
import java.util.*;
import zerf.aglet.*;

public class MutualExclusionAgent extends Aglet implements
MobilityListener
{
    private boolean Debug = true;
    private boolean experimentOn = true;
    private StationaryLogProxy meLog;
    private AgletInfo info;
    private String guideAgentProperty;
    private boolean busy = false;
    private AgletProxy locatorProxy;
    private AgletProxy guideProxy;
    private AgletProxy workerProxy;
    private Vector workerAgentNames = null;
    private Vector localMERRequests = null;
    private Vector locAgentMigrations = null;
    private int iTOTALMigrations = 0;
    private long lTotalWaitingTime = 0;
    private long lMinWaitingTime = 0;
    private long lMaxWaitingTime = 0;
    private int iTOTALPermissionsGranted = 0;
    private URL nextURL = null;

    public void onCreate(Object args)
    {
        addMobilityListener(this);
        info = getAgletInfo();
        localMERRequests = new Vector();
        locAgentMigrations = new Vector();
        workerAgentNames = new Vector();
        meLog =new StationaryLogProxy(getAgletContext());
        MEProperties oMEProperties = (MEProperties)args;
        guideAgentProperty = oMEProperties.getGuideAgentProperty();
        AgletContext ac = getAgletContext();
        AgletID aid = (AgletID) ac.getProperty(guideAgentProperty);
        if(aid != null)
        {
            guideProxy = ac.getAgletProxy(aid);
        }
        meLog.log("MEA: is created @ " +
            getAgletContext().getHostingURL());
    }

    public boolean handleMessage(Message msg)
    {
```

```

if(msg.sameKind("RELEASE"))
{
    String sWorkerAgentName = (String)((Object[])
        msg.getArg())[0];
    long lWaitingTime = ((Long)((Object[])
        msg.getArg())[1]).longValue();
    lTotalWaitingTime = lTotalWaitingTime + lWaitingTime;
    if(lWaitingTime < lMinWaitingTime)
    {
        lMinWaitingTime = lWaitingTime;
    }
    if(lWaitingTime > lMaxWaitingTime)
    {
        lMaxWaitingTime = lWaitingTime;
    }
    if(experimentOn)
    {
        meLog.log("*****
            *****");
        meLog.log("MEA: receive RELEAS from" +
            sWorkerAgentName);
        meLog.log("MEA: " + sWorkerAgentName + "Waited
            " + lWaitingTime);
        meLog.log("MEA: Total Waiting Time = " +
            lTotalWaitingTime);
        meLog.log("MEA: Total Permissions Granted = " +
            iTotPermissionsGranted);
        meLog.log("MEA: Total Permissions Granted = " +
            iTotPermissionsGranted);
        meLog.log("MEA: Average Waiting Time = " +
            lTotalWaitingTime/iTotPermissionsGranted);
        meLog.log("MEA: Minimum Waiting Time = " +
            lMinWaitingTime);
        meLog.log("MEA: Maximum Waiting Time = " +
            lMaxWaitingTime);
        meLog.log("MEA: Total Agent Migrations = " +
            iTotMigrations);
        meLog.log("MEA: Average Migrations = " +
            iTotMigrations/iTotPermissionsGranted);
        meLog.log("*****
            *****");
    }
    busy = false;
    processGrantRequest();
}
if (msg.sameKind("REQUEST"))
{
    locatorProxy = (AgletProxy)((Object[]) msg.getArg())[1];
    String sWorkerName = (String)((Object[])
        msg.getArg())[5];
    String sWorkerHome = (String)((Object[])
        msg.getArg())[4];
    int iLocatorMigrations = ((Integer)((Object[])
        msg.getArg())[6]).intValue();

```

```

iTotalMigrations = iTotalMigrations +
    iLocatorMigrations;
if (experimentOn)
{
    meLog.log("MEA: Received REQUEST From " +
        sWorkerName);
}
if (!(sWorkerHome.equals(getAgletContext()
    .getHostingURL().toString())))
{
    if (experimentOn)
    {
        meLog.log("MEA: REQUEST From " +
            sWorkerName + " is From Remote
            agent: " + sWorkerHome);
    }
    try
    {
        nextURL = new URL(sWorkerHome);
    }
    catch (Exception e)
    {
        if (experimentOn)
        {
            meLog.log("MEA: Exception in
                making URL");
            meLog.log(e);
        }
    }
}
else
{
    if (experimentOn)
    {
        meLog.log("MEA: request from local " +
            sWorkerName);
    }
    workerProxy = (AgletProxy) ((Object[])
        msg.getArg()[0]);
    localMERequests.add(workerProxy);
    workerAgentNames.add(sWorkerName);
    locAgentMigrations.add(new
        Integer(iLocatorMigrations));
}
if (experimentOn)
{
    meLog.log("MEA: sends DISPOSE message to LA of
        " + sWorkerName);
}
try
{
    locatorProxy.sendOnewayMessage(new
        Message("DISPOSE"));
}

```

```

        catch(Exception e)
        {
            if(experimentOn)
            {
                meLog.log("MEA: Exception in sending
                           DISPOSE message to LA");
                meLog.log(e);
            }
        }
        processGrantRequest();
        return true;
    }
    else
    {
        return false;
    }
}

public synchronized void processGrantRequest()
{
    if(!busy)
    {
        if(localMERRequests.size() != 0)
        {
            busy = true;
            if(experimentOn)
            {
                meLog.log("MEA: There are Pending Local
                           Requests");
                meLog.log("MEA: No. Of Pending Requests
                           = " + localMERRequests.size());
            }
            workerProxy =
                (AgletProxy) localMERRequests.remove(0);
            String sWorkerName =
                (String) workerAgentNames.remove(0);
            try
            {
                workerProxy.sendOnewayMessage(new
                    Message("GRANT", getProxy()));
                iTotalPermissionsGranted++;
                if(experimentOn)
                {
                    meLog.log("MEA: sends GRANT to
                               " + sWorkerName);
                }
            }
            catch(Exception e)
            {
                meLog.log("MEA Failde to send grants "
                           + sWorkerName);
                meLog.log(e);
            }
        }
    }
}

```

```

        }
        if(nextURL != null)
        {
            dispatchToNextURL();
        }
    }
}

private void dispatchToNextURL()
{
    try
    {
        if(Debug)
        {
            System.out.println("MEA is going to move to " +
                               nextURL.toString());
        }
        locatorProxy.sendOnewayMessage(new Message("DISPOSE"));
    }
    catch (Exception e)
    {
        meLog.log("MEA Exception in sending DISPOSE message: " +
                  e);
    }
    locatorProxy = null;
    workerProxy = null;
    localMERequests = null;
    workerAgentNames = null;
    locAgentMigrations = null;
    if(Debug)
    {
        meLog.log("MEA is going to move to " +
                  nextURL.toString());
    }
    try
    {
        dispatch(nextURL);
    }
    catch (Exception e)
    {
        meLog.log("MEA Exception in moving to : " +
                  nextURL.toString());
        meLog.log("MEA Except: " + e);
    }
}

private synchronized void sendArrivalMessage()
{
    try
    {
        if(experimentOn)
        {
            meLog.log("*****
*****");

```

```

        meLog.log("MEA: MEA_ARRIVAL to GA ");
    }
    Object oReply = guideProxy.sendMessage(new
        Message("ARRIVAL",getProxy()));
    localMERequests = (Vector)((Object[])oReply)[0];
    workerAgentNames = (Vector)((Object[])oReply)[1];
    locAgentMigrations = (Vector)((Object[])oReply)[2];
    String sURL = (String)((Object[])oReply)[3];
    int iLocatorMigrations=((Integer)((Object[])
        oReply)[4]).intValue();
    iTotMigrations = iTotMigrations +
        iLocatorMigrations;
    if(sURL != null)
    {
        nextURL = new URL(sURL);
    }
    if(experimentOn)
    {
        meLog.log("MEA: GA transfers following to MEA
            ");
        meLog.log("MEA: Vector of Worker Agents with
            size = " + localMERequests.size());
        meLog.log("MEA: Vector of Worker Agents Names
            with size = " + workerAgentNames.size());
        meLog.log("MEA: Vector of Lcator Agent
            Migrations with size = " +
            locAgentMigrations.size());
        meLog.log("MEA: Next URL with value = " +
            sURL);
        meLog.log("*****
            *****");
    }
}
catch (Exception e)
{
    System.out.println("MEA: Exceptipn in sending
        MEA_ARRIVAL to GA "+ e);
}
}

public void onArrival(MobilityEvent ev)
{
    if(Debug)
    {
        System.out.println("MEA arrived");
    }
    meLog.log("MEA: arrived @ " + getAgletContext().getHostingURL()
        );
    AgletContext ac = getAgletContext();
    AgletID aid = (AgletID) ac.getProperty(guideAgentProperty);
    if(aid != null)
    {
        guideProxy = ac.getAgletProxy(aid);
    }
}

```

```

        nextURL = null;
        sendArrivalMessage();
        busy = false;
        processGrantRequest();
    }

    public void onDispatching(MobilityEvent ev)
    {
        try
        {
            guideProxy.sendOnewayMessage(new Message("DEPARTURE"));
        }
        catch(Exception e)
        {
            System.out.println("MEA Exception: in sending DEPARTURE");
            System.out.println(e);
        }
    }

    public String toString()
    {
        StringBuffer sbGuideAgent = new StringBuffer();
        sbGuideAgent.append("MutualExclusionAgent State:");
        sbGuideAgent.append("\n");
        sbGuideAgent.append(info.toString()).append("\n");
        sbGuideAgent.append("busy: ").append(busy).append("\n");
        sbGuideAgent.append("nextURL: ").append(nextURL).append("\n");
        return sbGuideAgent.toString();
    }
}

```

Locator Agent

```

package fayyaz.mutualexclusion;

import com.ibm.aglet.*;
import com.ibm.aglet.event.*;

public class LocatorAgent extends Aglet implements MobilityListener
{
    private boolean Debug = true;
    private boolean experimentOn = true;
    private String name = null;
    private AgletInfo info = null;
    private String locatorHome = null;
    private String first_directionURL = null;
    private AgletProxy guideProxy = null;
    private AgletProxy workerProxy = null;
    private String guideAgentProperty = null;
    private int iMigrationCount;

    public void onCreation(Object args)
    {

```



```

addMobilityListener(this);
guideProxy = (AgletProxy)((Object[])args)[0];
workerProxy = (AgletProxy)((Object[])args)[1];
guideAgentProperty = (String)((Object[])args)[2];
locatorHome = getAgletContext().getHostingURL().toString();
first_directionURL =
    getAgletContext().getHostingURL().toString();
info = getAgletInfo();
name = (String)((Object[])args)[3];
if(Debug)
{
    System.out.println("\n\nLA: is created \n" + name);
    System.out.println("\n\nLA: is created \n" +
        toString());
}
iMigrationCount = 0;
sendMERequest();
}

public void onArrival(MobilityEvent ev)
{
    if(Debug)
    {
        System.out.println("\n\nLA just Arrived\n" +
            toString());
    }
    AgletContext ac = getAgletContext();
    AgletID aid = (AgletID) ac.getProperty(guideAgentProperty);
    if(aid != null)
    {
        guideProxy = ac.getAgletProxy(aid);
    }
    ++iMigrationCount;
    sendMERequest();
}

public void onDispatching(MobilityEvent ev)
{
    first_directionURL =
        getAgletContext().getHostingURL().toString();
}

private void sendMERequest()
{
    try
    {
        if(Debug)
        {
            System.out.println(name + " sends REQUEST to
                GA");
        }
        Object oArgsForGA = new Object[]
        {workerProxy,getProxy(),first_directionURL,locatorHome
            ,name,new Integer(iMigrationCount)};

```

```

        guideProxy.sendOnewayMessage(new
            Message("REQUEST",oArgsForGA));
    }
    catch(Exception e)
    {
        System.out.println("Exception LA:" + name + "
            Failed to send REQUEST to GA");
        System.out.println(e);
    }
}

public boolean handleMessage(Message msg)
{
    if(msg.sameKind("DISPOSE"))
    {
        if(Debug)
        {
            System.out.println(name + "LA: got a message
                DISPOSE from MEA");
            dispose();
        }
        return true;
    }
    else
    {
        return false;
    }
}

public String toString()
{
    StringBuffer sbGuideAgent = new StringBuffer();
    sbGuideAgent.append("\n\nLocatorAgent State: ").append("\n");
    sbGuideAgent.append(info.toString()).append("\n");
    sbGuideAgent.append("first_DirectionURL:
        ").append(first_directionURL).append("\n");
    return sbGuideAgent.toString();
}
}

```

Guide Agent

```

package fayyaz.mutualexclusion;

import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import java.net.*;
import java.util.*;

public class GuideAgent extends Aglet
{
    private boolean Debug = true;
    private boolean experimentOn = true;

```

```

private AgletInfo info = null;
private String primaryMEDirection = null;
private String secondaryMEDirection = null;
private boolean possessMEA = false;
private String nextURL = null;
private AgletProxy MEProxy = null;
private AgletProxy locatorProxy = null;
private Vector localMERequests = new Vector();
private Vector storedWorkerNames = new Vector();
int iTotalMigrations = 0;

public void onCreate(Object init)
{
    info = getAgletInfo();
    Object obj = init;
    MEProperties oMEProperties = (MEProperties)obj;
    primaryMEDirection = oMEProperties.getPrimaryDirection();
    if(oMEProperties.getBeginME())
    {
        try
        {
            MEProxy = getAgletContext().createAglet
                (null,"fayyaz.mutualexclusion.
                MutualExclusionAgent",oMEProperties);
            possessMEA = true;
        }
        catch (Exception e)
        {
            System.out.println("GA EXCEPTION: failed to
            create GA");
        }
    }
    if(Debug)
    {
        System.out.println("\n\nGA: is created \n" +
            toString());
    }
}

public boolean handleMessage(Message msg)
{
    if (msg.sameKind("REQUEST"))
    {
        if(Debug)
        {
            System.out.println("GA: Received REQUEST");
        }
        processGrantRequest(msg);
        return true;
    }
    if (msg.sameKind("DEPARTURE"))
    {
        if(Debug)
        {

```

```

        System.out.println("GA: Received DEPARTURE");
    }
    possessMEA = false;
    return true;
}
if (msg.sameKind("ARRIVAL"))
{
    MEProxy = (AgletProxy)msg.getArg();
    possessMEA = true;
    Object args = new Object[] {localMERequests,
        storedWorkerNames,storedMigrationCount,
        nextURL,new Integer(iTotalMigrations)};
    msg.sendReply(args);
    localMERequests = new Vector();
    storedWorkerNames = new Vector();
    storedMigrationCount = new Vector();
    iTotalMigrations = 0;
    nextURL = null;
    if(Debug)
    {
        System.out.println("\n\nGA: Receive MEA_ARRIVAL
            \n" + toString());
    }
    return true;
}
else
{
    return false;
}
}

public synchronized void processGrantRequest(Message msg)
{
    if(Debug)
    {
        System.out.println("GA: processing REQUEST");
    }
    AgletProxy locatorProxy = (AgletProxy) ((Object[])
        msg.getArg())[1];
    String sLocatorAgentHome = (String) ((Object[])
        msg.getArg())[4];
    String sGuideAgentHome =
        getAgletContext().getHostingURL().toString();
    if(Debug)
    {
        System.out.println("sLocatorAgentHome: " +
            sLocatorAgentHome );
        System.out.println("sGuideAgentHome: " +
            sGuideAgentHome);
    }
    if(primaryMEDirection.equals(getAgletContext().
        getHostingURL().toString()))
    {
        if(Debug)

```

```

{
    System.out.println("GA: do not Forward
                        REQUEST");
}
if (possessMEA)
{
    if (Debug)
    {
        System.out.println("GA: Server Possess
                            MEA");
    }
    try
    {
        if (Debug)
        {
            System.out.println("GA:
                                Forward LA_REQUEST to MEA");
        }
        MEProxy.sendOnewayMessage(msg);
    }
    catch (Exception e)
    {
        System.out.println("Exception GA:
                            Failed to forward LA_REQUEST to MEA");
        System.out.println(e);
    }
}
else
{
    if (Debug)
    {
        System.out.println("GA: does not
                            possess MEA");
        System.out.println("GA: Server has
                            requested MEA");
    }

    if (!(sLocatorAgentHome.equals
          (sGuideAgentHome)))
    {
        if (Debug)
        {
            System.out.println("GA:
                                REQUEST from remote Agent");
        }
        nextURL = (String) ((Object[])
                             msg.getArg())[4];
        int iLocatorMigrations
            = ((Integer) ((Object[])
                           msg.getArg())
              [6]).intValue();
        iTotalMigrations = iTotalMigrations +
                           iLocatorMigrations;
    }
    else

```

```

        {
            if (Debug)
            {
                System.out.println("GA:
                REQUEST from local Agent");
            }
            localMERequests.add((AgletProxy)
                ((Object[]) msg.getArg())[0]);

            storedWorkerNames.add((String)
                ((Object[]) msg.getArg())[5]);
            storedMigrationCount.add((Integer)
                ((Object[]) msg.getArg())[6]);

            if (Debug)
            {
                System.out.println("GA:
                REQUEST have been added");
                System.out.println("GA: Size
                Of storedWorkerNames = " +
                storedWorkerNames.size());
                System.out.println("GA: Size
                Of localMERequests = "
                + localMERequests.size());
                System.out.println("GA: Size
                Of storedMigrationCount = " +
                storedMigrationCount.size());
            }
        }
        if (Debug)
        {
            System.out.println("GA: sends DISPOSE
            message to LA");
        }
        try
        {
            locatorProxy.sendOnewayMessage(new
                Message("DISPOSE"));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
else
{
    if (Debug)
    {
        System.out.println("GA Dispatching LA to: " +
            primaryMEDirection);
    }
    if (sLocatorAgentHome.equals(sGuideAgentHome))

```

```

        {
            if (Debug)
            {
                System.out.println("GA: REQUEST from
                                   local Agent");
            }
            localMERequests.add((AgletProxy) ((Object[])
                                                msg.getArg()[0]));
            storedWorkerNames.add((String) ((Object[])
                                                msg.getArg()[5]));
            storedMigrationCount.add((Integer) ((Object[])
                                                msg.getArg()[6]));
        }
        try
        {
            locatorProxy.dispatch(new
                                   URL(primaryMEDirection));
        }
        catch (Exception e)
        {
            System.out.println("GA EXCEPTION: failed to
                               dispatch LA");
            System.out.println(e);
        }
    }
    /* change direction pointers */
    primaryMEDirection = (String) ((Object[]) msg.getArg()[2]);
}

public String toString()
{
    StringBuffer sbGuideAgent = new StringBuffer();
    sbGuideAgent.append("GuideAgent State: ").append("\n");
    sbGuideAgent.append(info.toString()).append("\n");
    sbGuideAgent.append("primaryMEDirection:
                        ").append(primaryMEDirection).append("\n");
    sbGuideAgent.append("nextURL: ").append(nextURL).append("\n");
    sbGuideAgent.append("possessMEA
                        ").append(possessMEA).append("\n");
    return sbGuideAgent.toString();
}
}

```

ME Properties

```

package fayyaz.mutualexclusion;

public class MEProperties implements java.io.Serializable
{
    private String guideAgentProperty;
    private String mEAgentProperty;
    private String primaryDirection;
    private boolean beginME;

```

```

public MEProperties ( String sGuideAgentProperty,
                     String sMEAgentProperty,
                     String sPrimaryDirection,
                     boolean bBeginME)
{
    guideAgentProperty = sGuideAgentProperty;
    mEAgentProperty = sMEAgentProperty;
    primaryDirection = sPrimaryDirection;
    beginME = bBeginME;
}

public void setGuideAgentProperty (String sGuideAgentProperty)
{
    guideAgentProperty = sGuideAgentProperty;
}

public String getGuideAgentProperty ()
{
    return guideAgentProperty;
}

public void setMEAgentProperty (String sMEAgentProperty)
{
    mEAgentProperty = sMEAgentProperty;
}

public String getMEAgentProperty ()
{
    return mEAgentProperty;
}

public void setPrimaryDirection (String sPrimaryDirection)
{
    primaryDirection = sPrimaryDirection;
}

public String getPrimaryDirection ()
{
    return primaryDirection;
}

public void setBeginME (boolean bBeginME)
{
    beginME = bBeginME;
}

public boolean getBeginME ()
{
    return beginME;
}

public String toString()
{

```



```
String temp = ("MEProperties:\n");  
temp = temp + ("Primary Direction: " + primaryDirection + "\n");  
return temp;  
    }  
}
```

APPENDIX-II

Sample Execution Trace

1051490246114 MEA: is created @ atp://ustad:10000/
 1051490246124 MEA: Received REQUEST From W_1051490245764
 1051490246124 MEA: request from local W_1051490245764
 1051490246124 MEA: sends DISPOSE message to LA of W_1051490245764
 1051490246164 MEA: There are Pending Local Requests
 1051490246174 MEA: No. Of Pending Requests = 1
 1051490246885 MEA: sends GRANT to W_1051490245764
 1051490246885 *****
 1051490246895 MEA: receive RELEAS fromW_1051490245764
 1051490246895 MEA: W_1051490245764Waited 240
 1051490246895 MEA: Total Waiting Time = 240
 1051490246895 MEA: Total Permissions Granted = 1
 1051490246905 MEA: Total Permissions Granted = 1
 1051490246905 MEA: Average Waiting Time = 240
 1051490246905 MEA: Minimum Waiting Time = 240
 1051490246905 MEA: Maximum Waiting Time = 240
 1051490246915 MEA: Total Agent Migrations = 0
 1051490246925 MEA: Average Migrations = 0
 1051490246925 *****
 1051490249098 MEA: Received REQUEST From W_1051490249088
 1051490249108 MEA: request from local W_1051490249088
 1051490249108 MEA: sends DISPOSE message to LA of W_1051490249088
 1051490249118 MEA: There are Pending Local Requests
 1051490249118 MEA: No. Of Pending Requests = 1
 1051490249118 MEA: sends GRANT to W_1051490249088
 1051490249128 *****
 1051490249128 MEA: receive RELEAS fromW_1051490249088
 1051490249128 MEA: W_1051490249088Waited 30
 1051490249139 MEA: Total Waiting Time = 270
 1051490249139 MEA: Total Permissions Granted = 2
 1051490249139 MEA: Total Permissions Granted = 2
 1051490249139 MEA: Average Waiting Time = 135
 1051490249149 MEA: Minimum Waiting Time = 30
 1051490249149 MEA: Maximum Waiting Time = 240
 1051490249159 MEA: Total Agent Migrations = 0
 1051490249159 MEA: Average Migrations = 0
 1051490249159 *****
 1051490250170 MEA: Received REQUEST From W_1051490245764
 1051490250170 MEA: REQUEST From W_1051490245764 is From Remote agent: atp://ustad:10003/
 1051490250190 MEA: sends DISPOSE message to LA of W_1051490245764
 1051490250200 MEA is going to move to atp://ustad:10003/
 1051490250360 MEA: arrived @ atp://ustad:10003/
 1051490250390 *****
 1051490250410 MEA: MEA_ARRIVAL to GA
 1051490250440 MEA: GA transfers following to MEA
 1051490250450 MEA: Vector of Worker Agents with size = 2
 1051490250470 MEA: Vector of Worker Agents Names with size = 2
 1051490250490 MEA: Vector of Lcator Agent Migrations with size = 2
 1051490250510 MEA: Next URL with value = null
 1051490250531 *****
 1051490250541 MEA: There are Pending Local Requests
 1051490250561 MEA: No. Of Pending Requests = 2
 1051490250581 MEA: sends GRANT to W_1051490245764
 1051490250611 *****
 1051490250631 MEA: receive RELEAS fromW_1051490245764
 1051490250651 MEA: W_1051490245764Waited 3636
 1051490250681 MEA: Total Waiting Time = 3906
 1051490251151 MEA: Total Permissions Granted = 3
 1051490251272 MEA: Total Permissions Granted = 3
 1051490251292 MEA: Average Waiting Time = 1302
 1051490251412 MEA: Minimum Waiting Time = 30
 1051490251442 MEA: Maximum Waiting Time = 3636
 1051490251452 MEA: Total Agent Migrations = 1

```

1051490251472 MEA: Average Migrations = 0
1051490251492 *****
1051490251542 MEA: There are Pending Local Requests
1051490251562 MEA: No. Of Pending Requests = 1
1051490251582 MEA: sends GRANT to W_1051490249088
1051490251622 *****
1051490251662 MEA: receive RELEAS fromW_1051490249088
1051490251692 MEA: W_1051490249088Waited 2363
1051490251712 MEA: Total Waiting Time = 6269
1051490251742 MEA: Total Permissions Granted = 4
1051490251832 MEA: Total Permissions Granted = 4
1051490251872 MEA: Average Waiting Time = 1567
1051490251912 MEA: Minimum Waiting Time = 30
1051490251933 MEA: Maximum Waiting Time = 3636
1051490251953 MEA: Total Agent Migrations = 1
1051490251973 MEA: Average Migrations = 0
1051490252003 *****
1051490254436 MEA: Received REQUEST From W_1051490251242
1051490254456 MEA: REQUEST From W_1051490251242 is From Remote agent: atp://ustad:10000/
1051490254486 MEA: sends DISPOSE message to LA of W_1051490251242
1051490254496 MEA is going to move to atp://ustad:10000/
1051490254596 MEA: arrived @ atp://ustad:10000/
1051490254606 *****
1051490254606 MEA: MEA_ARRIVAL to GA
1051490254616 MEA: GA transfers following to MEA
1051490254616 MEA: Vector of Worker Agents with size = 1
1051490254626 MEA: Vector of Worker Agents Names with size = 1
1051490254626 MEA: Vector of Lcator Agent Migrations with size = 1
1051490254626 MEA: Next URL with value = null
1051490254636 *****
1051490254636 MEA: There are Pending Local Requests
1051490254646 MEA: No. Of Pending Requests = 1
1051490255137 MEA: sends GRANT to W_1051490251242
1051490255147 *****
1051490255147 MEA: receive RELEAS fromW_1051490251242
1051490255157 MEA: W_1051490251242Waited 3344
1051490255157 MEA: Total Waiting Time = 9613
1051490255167 MEA: Total Permissions Granted = 5
1051490255167 MEA: Total Permissions Granted = 5
1051490255177 MEA: Average Waiting Time = 1922
1051490255177 MEA: Minimum Waiting Time = 30
1051490255187 MEA: Maximum Waiting Time = 3636
1051490255187 MEA: Total Agent Migrations = 2
1051490255197 MEA: Average Migrations = 0
1051490255197 *****
1051490258332 MEA: Received REQUEST From W_1051490245764
1051490258332 MEA: REQUEST From W_1051490245764 is From Remote agent: atp://ustad:10001/
1051490258342 MEA: sends DISPOSE message to LA of W_1051490245764
1051490258392 MEA Exception in sending DISPOSE message: com.ibm.aglet.InvalidAgletException: Aglet is not
valid
1051490258402 MEA is going to move to atp://ustad:10001/
1051490258572 MEA: arrived @ atp://ustad:10001/
1051490258602 *****
1051490258622 MEA: MEA_ARRIVAL to GA
1051490258642 MEA: GA transfers following to MEA
1051490258662 MEA: Vector of Worker Agents with size = 2
1051490258672 MEA: Vector of Worker Agents Names with size = 2
1051490258692 MEA: Vector of Lcator Agent Migrations with size = 2
1051490258722 MEA: Next URL with value = null
1051490258742 *****
1051490258752 MEA: There are Pending Local Requests
1051490258772 MEA: No. Of Pending Requests = 2
1051490258792 MEA: sends GRANT to W_1051490245764

```

```

1051490258812 *****
1051490258842 MEA: receive RELEAS fromW_1051490245764
1051490258873 MEA: W_1051490245764Waited 7370
1051490259333 MEA: Total Waiting Time = 16983
1051490259353 MEA: Total Permissions Granted = 6
1051490259373 MEA: Total Permissions Granted = 6
1051490259393 MEA: Average Waiting Time = 2830
1051490259423 MEA: Minimum Waiting Time = 30
1051490259463 MEA: Maximum Waiting Time = 7370
1051490259493 MEA: Total Agent Migrations = 4
1051490259513 MEA: Average Migrations = 0
1051490259543 *****
1051490259574 MEA: There are Pending Local Requests
1051490259594 MEA: No. Of Pending Requests = 1
1051490259624 MEA: sends GRANT to W_1051490249088
1051490259684 *****
1051490259744 MEA: receive RELEAS fromW_1051490249088
1051490259774 MEA: W_1051490249088Waited 7802
1051490259804 MEA: Total Waiting Time = 24785
1051490259854 MEA: Total Permissions Granted = 7
1051490259884 MEA: Total Permissions Granted = 7
1051490259924 MEA: Average Waiting Time = 3540
1051490259944 MEA: Minimum Waiting Time = 30
1051490259964 MEA: Maximum Waiting Time = 7802
1051490260034 MEA: Total Agent Migrations = 4
1051490260054 MEA: Average Migrations = 0
1051490260074 *****
1051490263059 MEA: Received REQUEST From W_1051490245764
1051490263079 MEA: REQUEST From W_1051490245764 is From Remote agent: atp://ustad:10004/
1051490263099 MEA: sends DISPOSE message to LA of W_1051490245764
1051490263119 MEA is going to move to atp://ustad:10004/
1051490263259 MEA: arrived @ atp://ustad:10004/
1051490263289 *****
1051490263309 MEA: MEA_ARRIVAL to GA
1051490263329 MEA: GA transfers following to MEA
1051490263349 MEA: Vector of Worker Agents with size = 1
1051490263379 MEA: Vector of Worker Agents Names with size = 1
1051490263399 MEA: Vector of Lcator Agent Migrations with size = 1
1051490263419 MEA: Next URL with value = null
1051490263439 *****
1051490263459 MEA: There are Pending Local Requests
1051490263479 MEA: No. Of Pending Requests = 1
1051490263499 MEA: sends GRANT to W_1051490245764
1051490263519 *****
1051490263539 MEA: receive RELEAS fromW_1051490245764
1051490263559 MEA: W_1051490245764Waited 3936
1051490263589 MEA: Total Waiting Time = 28721
1051490263609 MEA: Total Permissions Granted = 8
1051490263669 MEA: Total Permissions Granted = 8
1051490263699 MEA: Average Waiting Time = 3590
1051490263719 MEA: Minimum Waiting Time = 30
1051490263750 MEA: Maximum Waiting Time = 7802
1051490263770 MEA: Total Agent Migrations = 5
1051490263800 MEA: Average Migrations = 0
1051490263820 *****
1051490267695 MEA: Received REQUEST From W_1051490251242
1051490267725 MEA: REQUEST From W_1051490251242 is From Remote agent: atp://ustad:10002/
1051490267745 MEA: sends DISPOSE message to LA of W_1051490251242
1051490267775 MEA is going to move to atp://ustad:10002/
1051490267926 MEA: arrived @ atp://ustad:10002/
1051490267946 *****
1051490267986 MEA: MEA_ARRIVAL to GA
1051490268006 MEA: GA transfers following to MEA

```

1051490268026 MEA: Vector of Worker Agents with size = 3
 1051490268046 MEA: Vector of Worker Agents Names with size = 3
 1051490268066 MEA: Vector of Lcator Agent Migrations with size = 3
 1051490268096 MEA: Next URL with value = null
 1051490268116 *****
 1051490268136 MEA: There are Pending Local Requests
 1051490268156 MEA: No. Of Pending Requests = 3
 1051490268176 MEA: sends GRANT to W_1051490251242
 1051490268196 *****
 1051490268236 MEA: receive RELEAS fromW_1051490251242
 1051490268256 MEA: W_1051490251242Waited 12909
 1051490268276 MEA: Total Waiting Time = 41630
 1051490268296 MEA: Total Permissions Granted = 9
 1051490268316 MEA: Total Permissions Granted = 9
 1051490268346 MEA: Average Waiting Time = 4625
 1051490268406 MEA: Minimum Waiting Time = 30
 1051490268426 MEA: Maximum Waiting Time = 12909
 1051490268446 MEA: Total Agent Migrations = 9
 1051490268486 MEA: Average Migrations = 1
 1051490268506 *****
 1051490268526 MEA: There are Pending Local Requests
 1051490268556 MEA: No. Of Pending Requests = 2
 1051490268586 MEA: sends GRANT to W_1051490249088
 1051490268606 *****
 1051490268637 MEA: receive RELEAS fromW_1051490249088
 1051490268667 MEA: W_1051490249088Waited 8752
 1051490268687 MEA: Total Waiting Time = 50382
 1051490268707 MEA: Total Permissions Granted = 10
 1051490268737 MEA: Total Permissions Granted = 10
 1051490268757 MEA: Average Waiting Time = 5038
 1051490268817 MEA: Minimum Waiting Time = 30
 1051490268887 MEA: Maximum Waiting Time = 12909
 1051490268917 MEA: Total Agent Migrations = 9
 1051490268937 MEA: Average Migrations = 0
 1051490268957 *****
 1051490268987 MEA: There are Pending Local Requests
 1051490269007 MEA: No. Of Pending Requests = 1
 1051490269037 MEA: sends GRANT to W_1051490245764
 1051490269067 *****
 1051490269087 MEA: receive RELEAS fromW_1051490245764
 1051490269117 MEA: W_1051490245764Waited 5318
 1051490269157 MEA: Total Waiting Time = 55700
 1051490269187 MEA: Total Permissions Granted = 11
 1051490269207 MEA: Total Permissions Granted = 11
 1051490269227 MEA: Average Waiting Time = 5063
 1051490269257 MEA: Minimum Waiting Time = 30
 1051490269287 MEA: Maximum Waiting Time = 12909
 1051490269338 MEA: Total Agent Migrations = 9
 1051490269368 MEA: Average Migrations = 0
 1051490269388 *****
 1051490278040 MEA: Received REQUEST From W_1051490251242
 1051490278060 MEA: REQUEST From W_1051490251242 is From Remote agent: atp://ustad:10001/
 1051490278080 MEA: sends DISPOSE message to LA of W_1051490251242
 1051490278110 MEA is going to move to atp://ustad:10001/
 1051490278180 MEA: arrived @ atp://ustad:10001/
 1051490278210 *****
 1051490278230 MEA: MEA_ARRIVAL to GA
 1051490278270 MEA: GA transfers following to MEA
 1051490278290 MEA: Vector of Worker Agents with size = 1
 1051490278320 MEA: Vector of Worker Agents Names with size = 1
 1051490278340 MEA: Vector of Lcator Agent Migrations with size = 1
 1051490278361 MEA: Next URL with value = atp://ustad:10004/
 1051490278381 *****

1051490278421 MEA: There are Pending Local Requests
 1051490278441 MEA: No. Of Pending Requests = 1
 1051490278461 MEA: sends GRANT to W_1051490251242
 1051490278491 MEA Exception in sending DISPOSE message: java.lang.NullPointerException
 1051490278521 MEA is going to move to atp://ustad:10004/
 1051490278691 MEA: arrived @ atp://ustad:10004/
 1051490278711 *****
 1051490278741 MEA: MEA_ARRIVAL to GA
 1051490278761 MEA: GA transfers following to MEA
 1051490278791 MEA: Vector of Worker Agents with size = 1
 1051490278821 MEA: Vector of Worker Agents Names with size = 1
 1051490278841 MEA: Vector of Lcator Agent Migrations with size = 1
 1051490278871 MEA: Next URL with value = null
 1051490278891 *****
 1051490278921 MEA: There are Pending Local Requests
 1051490278941 MEA: No. Of Pending Requests = 1
 1051490278971 MEA: sends GRANT to W_1051490249088
 1051490278991 *****
 1051490279021 MEA: receive RELEAS fromW_1051490249088
 1051490279052 MEA: W_1051490249088Waited 9914
 1051490279082 MEA: Total Waiting Time = 65614
 1051490279102 MEA: Total Permissions Granted = 13
 1051490279132 MEA: Total Permissions Granted = 13
 1051490279162 MEA: Average Waiting Time = 5047
 1051490279192 MEA: Minimum Waiting Time = 30
 1051490279222 MEA: Maximum Waiting Time = 12909
 1051490279242 MEA: Total Agent Migrations = 13
 1051490279272 MEA: Average Migrations = 1
 1051490279302 *****
 1051490287794 MEA: Received REQUEST From W_1051490251242
 1051490287824 MEA: REQUEST From W_1051490251242 is From Remote agent: atp://ustad:10000/
 1051490287854 MEA: sends DISPOSE message to LA of W_1051490251242
 1051490287884 MEA is going to move to atp://ustad:10000/
 1051490287954 MEA: arrived @ atp://ustad:10000/
 1051490287974 *****
 1051490287984 MEA: MEA_ARRIVAL to GA
 1051490287994 MEA: GA transfers following to MEA
 1051490288014 MEA: Vector of Worker Agents with size = 1
 1051490288024 MEA: Vector of Worker Agents Names with size = 1
 1051490288034 MEA: Vector of Lcator Agent Migrations with size = 1
 1051490288054 MEA: Next URL with value = null
 1051490288064 *****
 1051490288074 MEA: There are Pending Local Requests
 1051490288095 MEA: No. Of Pending Requests = 1
 1051490288145 MEA: sends GRANT to W_1051490251242
 1051490288155 *****
 1051490288175 MEA: receive RELEAS fromW_1051490251242
 1051490288195 MEA: W_1051490251242Waited 9484
 1051490288205 MEA: Total Waiting Time = 75098
 1051490288215 MEA: Total Permissions Granted = 14
 1051490288235 MEA: Total Permissions Granted = 14
 1051490288245 MEA: Average Waiting Time = 5364
 1051490288265 MEA: Minimum Waiting Time = 30
 1051490288275 MEA: Maximum Waiting Time = 12909
 1051490288295 MEA: Total Agent Migrations = 16
 1051490288305 MEA: Average Migrations = 1
 1051490577601 MEA: is created @ atp://ustad:10000/
 1051490577601 MEA: Received REQUEST From W_1051490577300
 1051490577601 MEA: request from local W_1051490577300
 1051490577611 MEA: sends DISPOSE message to LA of W_1051490577300
 1051490577611 MEA: There are Pending Local Requests
 1051490577611 MEA: No. Of Pending Requests = 1
 1051490577651 MEA: sends GRANT to W_1051490577300

```

1051490577651 *****
1051490577661 MEA: receive RELEAS fromW_1051490577300
1051490577661 MEA: W_1051490577300Waited 230
1051490577661 MEA: Total Waiting Time = 230
1051490577661 MEA: Total Permissions Granted = 1
1051490577661 MEA: Total Permissions Granted = 1
1051490577671 MEA: Average Waiting Time = 230
1051490577671 MEA: Minimum Waiting Time = 230
1051490577681 MEA: Maximum Waiting Time = 230
1051490577681 MEA: Total Agent Migrations = 0
1051490577681 MEA: Average Migrations = 0
1051490577681 *****
1051490579453 MEA: Received REQUEST From W_1051490579423
1051490579453 MEA: request from local W_1051490579423
1051490579463 MEA: sends DISPOSE message to LA of W_1051490579423
1051490579463 MEA: There are Pending Local Requests
1051490579474 MEA: No. Of Pending Requests = 1
1051490579474 MEA: sends GRANT to W_1051490579423
1051490579474 *****
1051490579474 MEA: receive RELEAS fromW_1051490579423
1051490579484 MEA: W_1051490579423Waited 31
1051490579484 MEA: Total Waiting Time = 261
1051490579484 MEA: Total Permissions Granted = 2
1051490579484 MEA: Total Permissions Granted = 2
1051490579494 MEA: Average Waiting Time = 130
1051490579504 MEA: Minimum Waiting Time = 31
1051490579504 MEA: Maximum Waiting Time = 230
1051490579534 MEA: Total Agent Migrations = 0
1051490579554 MEA: Average Migrations = 0
1051490579554 *****
1051490580775 MEA: Received REQUEST From W_1051490577300
1051490580775 MEA: REQUEST From W_1051490577300 is From Remote agent: atp://ustad:10004/
1051490580775 MEA: sends DISPOSE message to LA of W_1051490577300
1051490580795 MEA is going to move to atp://ustad:10004/
1051490580876 MEA: arrived @ atp://ustad:10004/
1051490580896 *****
1051490580916 MEA: MEA_ARRIVAL to GA
1051490580936 MEA: GA transfers following to MEA
1051490580956 MEA: Vector of Worker Agents with size = 1
1051490580966 MEA: Vector of Worker Agents Names with size = 1
1051490580986 MEA: Vector of Lcator Agent Migrations with size = 1

```


BIBLIOGRAPHY

- [1] Michael J. Dammer and Maurice P. Herlihy. The Arrow Distributed Directory Protocol. *Proceedings of DISC 1998*, pp 119-133, September 1998
- [2] M. Dillencourt, L. F. Bic, and M. Fukuda. Distributed computing using autonomous agents. *IEEE Computer*, Vol. 29, No. 8, pp 55-61, August 1996.
- [3] R. Gray, G. Cybenko, D. Kotz, and D. Rus. Mobile agents: Motivations and state of the art. Technical Report TR2000-365, Department of Computer Science, Dartmouth College, 2000.
- [4] D.B. Lange and M. Oshima. Programming and Deploying Java Mobile Agents with Aglets. Addison-Wesley, 1998.
- [5] Qusay H. Mahmoud. Distributed Programming with Java. Manning, 1999.
- [6] S. Mishra and P. Xie. Interagent Communication and Synchronization in DaAgent. *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems*, pp 699-702, Phoenix, AZ, April 2001
- [7] Mukesh Singhal. Taxonomy of distributed Mutual Exclusion. *Journal of Parallel and Distributed Computing*, Vol. 18, pp 94-101, June 1993.
- [8] Jiannong Cao, Xianbing Wang, and Jie Wu. A Mobile Agent Enabled Fully Distributed Mutual Exclusion Algorithm. *Proceedings of 6th International Conference, MA 2002*, pp 138-153, Barcelona, Spain, October , 2002.
- [9] Michael N. Huhns. Agent Teams: Building and implementing Software. *IEEE Internet Computing*, Vol. 4, No. 1, pp 93-95, February 2000.
- [10] Vera Nagamuta. Coordinating Mobile Agents through the Broadcast Channel, Masters Thesis. IME-University of Sao Paulo, Brazil, 2000
- [11] Leon Erlanger. Distributed Computin : An Introduction.
<http://www.extremetech.com/article2>. April, 2000
- [12] Martin Shepherd Chidyausiku. Efficient Protocols for distributed Mutual Exclusion and Termination Detectio. Masters Thesis, Carleton University, Ottawa, Canada. 1993
- [13] H. Nwana. Software Agents: An Overview. *Knowledge Engineering Review*, Vol 11, No. 3, pp 205-244, September 1996.

- [14] Joachim Baumann. Mobile Agents: Control Algorithms. Springer, Lecture Notes in Computer Science 1658, June 2000.
- [15] Bruce Ashfield. Distributed Deadlock Detection immobile Agent Systems. Masters Thesis, Carleton University, Ottawa, Canada. 2001
- [16] Shawn Green. Software Agents: A Review. Dublin: Department of Computer Science, Trinity College, 1997.
- [17] D. Milojicic, F. Douglass and R Wheeler. Mobility: Processes, Computers and Agents. Addison-Wesley, Reading, 1999.
- [18] S. Franklin and A. Graesser. Is it an Agent or just a Program?: A taxonomy for Autonomous Agents. Springer-Verlag, 1996.
- [19] E.W. Dijkstra. Programming Languages: Cooperating Sequential Process. Academic Press, New York, 1965.
- [20] K. Raymond. A tree based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, Vol 7, No. 1, pp 61-77, February 1989.
- [21] Y. I. Chang, M. Singhal and M. Liu. A fault tolerant algorithm for distributed mutual exclusion. *Proceedings of the 9th Symposium on Reliable Distributed Systems*, pp 146-154, 1990.
- [22] M. Singhal. A dynamic information structure mutual exclusion algorithm for distributed systems. *IEEE transactions on Parallel distributed Systems*, Vol. 3, No. 1, pp 121-125, 1992.
- [23] G. Rictal and A.K. Agrawala. An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM*, Vol. 24, No. 1, pp 9-17, January 1981.
- [24] M. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, Vol. 3, No. 2, pp 145-159, May 1985.
- [25] Y. I. Chang, M. Singhal and M. Liu. A hybrid algorithm for distributed mutual exclusion. *Proceedings of COMPSAC'90*, pp 10-12, October 1990.
- [26] G. Cabri, L Leonardi and F. Zambonelli. Mobile-Agent Coordination Models for Internet Applications. *IEEE Computer*, Vol. 33, No. 2. pp 82-89, February 2000.
- [27] Vu Anh Pham and Ahmed Karmouch. Mobile Software Agents: An Overview. *IEEE communication Magazine*, Vol. 36, No. 7, pp 26-37, July 1998.
- [28] <http://www.trl.ibm.com/aglets>.
- [29] <http://www.cs.uit.no/DOS/Tacoma/index.html>.

- [30] <http://www.objectspace.com/voyager>.
- [31] <http://www.meitca.com/HSL/Projects/Concordia>
- [32] <http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html>.
- [33] <http://www.generalmagic.com/technology/odyssey.html>.
- [34] L. Lamport, N. Lynch. Chapter-18, Hand book of Theoretical computer Science, Volume B, Formal Models and Semantics. The MIT Press, 1990.
- [35] C. Hoare. Communicating sequential processes. *Communications. ACM*, Vol. 2, No. 8, pp 666-677, August 1978.
- [36] P. Jalote. Fault Tolerance in Distributed Systems. PTR Prentice Hall, 1994
- [37] http://agent.csd.auth.gr/~zerf/StationaryLog/stationarylog_aglet.html