INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

MOBILE AGENTS IN DISTRIBUTED SEARCH: A COMPARATIVE STUDY

By

Elio Velazquez

A Thesis Submitted to
the Faculty of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the Degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario, Canada
January 2001

© Copyright Elio Velazquez, 2001

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

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.

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 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.

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.

0-612-57761-9

Canada

The undersigned hereby recommended to
Faculty of Graduate Studies and Research
acceptance of this thesis

MOBILE AGENTS IN DISTRIBUTED SEARCH: A COMPARATIVE STUDY

Submitted by Elio Velazquez, B.C.S.
in partial fulfillment of the requirements
for the degree of Master of Computer Science

Department Chair

Thesis Supervisor, Professor Nicola Santoro

Thesis Co-supervisor, Dr. Amiya Nayak

Carleton University

January 2001

ABSTRACT

This work evaluates the suitability of using mobile agent computing to solve a distributed search problem as an alternative to a message passing approach. The search problem is analyzed from two different perspectives, the single travelling agent case and the multiple travelling agents.

When decomposing a task into several independent sub-tasks, agents need strategies to coordinate their actions. This work also examines several coordination models as well as their implementation using agent patterns.

We provide an architecture to implement a distributed search which comprises single and multiple agent solutions as well as components to guarantee agent coordination. Several experiments are carried out to measure the performance of the mobile agent solutions as well as the stationary agents for different network topologies. The multiple agents approach shows better performance when the number of nodes in the network increases. The tree topology proposed provides the best overall performance among all topologies considered.

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisors Dr. Nicola Santoro and Dr. Amiya Nayak for their guidance, encouragement and financial support throughout this research. It has been a very rewarding experience to work with them.

I wish to express my most sincere gratitude to Mr. Roy Gibbons for his help since the beginning and for always being there when I needed it. Very special thanks to Ms. Janet King for her time and valuable suggestions.

I would like to thank Mr. Keith Hadley and Mr. Malcolm McDowell of Nortel Networks for supporting this research and for their constant encouragement.

And last but not least, I would like to express my deepest gratitude to my wife Yoli, my parents and friends.

TABLE OF CONTENTS

ABSTRACT.....	iii
ACKNOWLEDGEMENTS.....	iv
LIST OF FIGURES	viii
LIST OF TABLES.....	ix
Chapter 1 Introduction	1
1.1 Overview	1
1.2 Previous Work.....	3
1.3 Contributions of The Thesis.....	5
1.4 Thesis Outline	6
Chapter 2 Agents and Mobile Agent Systems.....	8
2.1 Introduction	8
2.2 Mobile Agent Platforms.....	12
2.2.1 Aglet.....	12
2.2.2 D`Agents	13
2.2.3 Concordia	13
2.3 Ara: Agents for Remote Access.....	14
2.4 TACOMA.....	17

Chapter 3 Distributed Search and Mobile Agent Computing.....	22
3.1 Introduction	22
3.2 The Single Travelling Agent.....	24
3.3 Flooding the Network.....	27
Chapter 4 Coordination Models in Distributed Search.....	31
4.1 Introduction	31
4.2 Various Coordination Models.....	31
4.2.1 Direct Coordination Model	32
4.2.2 Meeting Oriented Coordination Model	33
4.2.3 Whiteboards, Tuples and Linda-like Coordination Models.....	33
4.3 Initiation of Agent Interactions	36
4.4 Proposed Coordination Model for Distributed Search.....	38
4.4.1 Whiteboard Coordination.....	38
4.4.2 The Agent Router	40
Chapter 5 Agent and Message Based Solutions	44
5.1 Introduction	44
5.2 The Agent Based Algorithms.....	45
5.2.1 The Single Travelling Agent.....	47
5.2.2 Flooding Algorithm for Mobile Agents	48
5.3 The Message Passing Algorithms	49
5.3.1 The Single Message Algorithm.....	50
5.3.2 The Flooding Algorithm For Stationary Agents	50

Chapter 6 Implementation Issues and Experimental Results	52
6.1 Introduction	52
6.2 Performance Issues.....	52
6.3 Detailed Implementation.....	55
6.4 Experimental Results.....	57
6.4.1 Test Case 1: Variable Network Size	58
6.4.2 Test Case 2: Variable Number of Neighbours	61
6.4.3 Test Case 3: Different Topologies	63
6.5 Summary of Experimental Results.....	67
Chapter 7 Conclusions	69
7.1 Thesis Summary.....	69
7.2 Future Work	71
References.....	74
Appendix A Flow Charts	78
Appendix B Test Results	82

LIST OF FIGURES

Figure 2.3.1 The Ara system architecture [PS97]. Example for languages A and β ..	15
Figure 2.4.1. Briefcases and Cabinets	18
Figure 3.1.1 Logical Spanning Tree Mapping.	23
Figure 3.2.1. Modifying the HOST folder to implement traversal algorithms.	26
Figure 4.4.1 The Whiteboard Pattern [DW99].....	39
Figure 4.4.2 The Router Pattern [DW99].....	42
Figure 6.4.1. Average execution times for mobile agents.....	59
Figure 6.4.2. Average execution times for stationary agents.	59
Figure 6.4.2 Average execution times for multiple travelling agents.	65
Figure 6.4.4 Average execution times for multiple messages.	66
Figure 7.1.1 System Architecture.....	70
Figure A.1 The Single Travelling Agent.....	78
Figure A.2 The Single Travelling Message	79
Figure A.3 Multiple Travelling Agents.....	80
Figure A.4 The Multiple Travelling Messages	81

LIST OF TABLES

Table 1: Variable Network Size ($N=3$)	82
Table 2: Variable Network Size ($N=6$)	83
Table 3: Variable Network Size ($N=9$)	84
Table 4: Variable Network Size ($N=12$)	85
Table 5: Variable Network Size ($N=15$)	86
Table 6: Variable Network Size ($N=18$)	87
Table 7: Variable Network Size ($N=21$)	88
Table 8: Variable Network Size ($N=24$)	89
Table 9: Variable Network Size ($N=27$)	90
Table 10: Variable Network Size ($N=30$)	91
Table 11: Variable Number of Neighbours ($k=2$)	92
Table 12: Variable Number of Neighbours ($K=4$)	93
Table 13: Variable Number of Neighbours ($K=6$)	94
Table 14: Variable Number of Neighbours ($K=8$)	95
Table 15: Variable Number of Neighbours ($K=10$)	96
Table 16: Variable Number of Neighbours ($K=12$)	97
Table 17: Variable Number of Neighbours ($K=14$)	98
Table 18: Variable Number of Neighbours ($K=16$)	99
Table 19: Variable Number of Neighbours ($K=19$)	100
Table 20: Bi-directional Ring ($N=16$)	101
Table 21: Unidirectional Ring ($N=16$)	102
Table 22: Hypercube ($N=16$)	103

Chapter 1 Introduction

1.1 Overview

This thesis examines the subject of mobile agent computing and its application in a distributed computing environment as an alternative to traditional message passing architecture. In particular, the distributed search problem is discussed along with some agent-based solutions.

Mobile agent computing and mobile agent architectures are trendy nowadays with efforts not only at the application level but also at the operating system level. The AgentOS project is an example of an agent-based operating system [L98]. Mobile agent computing, seen as a special case of message passing, intends to move computations as close as possible to the data. It also claims to make a more efficient use of the bandwidth by considerably decreasing the number of messages exchanged between cooperating applications.

Agents are programs that help accomplish a task without a continued interaction of the user. Agents can socialize and interact with other agents. They can be pro-active or reactive and more importantly, stationary or mobile. Mobile agents are the focus of this thesis although some auxiliary stationary agents will be also discussed. This work will analyse particular mobile agent approaches for the distributed search problem.

Mobile agents by themselves cannot exist in a networked environment without a platform that guarantees the most important and distinctive property: migration. Mobile agent systems or mobile agent platforms offer an environment for agent execution. They are responsible for executing the agent, sending the agent across the network and reactivating incoming agents. There are several mobile agents platforms available but some of them might pose an excessive overhead if deployed as an underlying mechanism in a distributed computing environment. Throughout this thesis, the suitability of this approach is evaluated based on a specific agent platform and focussing on the distributed search problem.

How many agents do we need to perform a distributed search or information retrieval? This work attempts to provide an answer to this question by comparing single agent approaches with multiple agent approaches. Multi agent systems seem very useful when solving any distributed problem. A "divide and conquer" approach can also be applied to mobile agents. By injecting several agents into the network, an application can decompose its task into several sub-tasks and devote one agent to each specific sub-task. The division of responsibilities among several agents introduces a new problem. These agents need to communicate and coordinate their actions in order to avoid duplicating efforts. Agent coordination models and agent patterns play an important role in the process of designing a mobile agent solution. This thesis also examines some characteristics of the current coordination models and proposes a formula, which combines some of the models into a particular solution.

1.2 Previous Work

The performance of mobile agents as well as their applicability as an alternative to the client/server technology has been studied in other application domains. The applicability of mobile agents for network management tasks, in particular for the management of MIBs (management information base) is examined in [OK99]. The experiments are carried out with client/server applications and mobile agents implemented in Java. The purpose of the experiments is to demonstrate the feasibility of using mobile agents to update the network nodes MIBs. It is concluded that, if the main objective is to decrease the bandwidth of a network with a significant number of nodes, it is preferable to use mobile agents to update MIBs in the network. By employing a mobile agent approach, the administrator no longer needs to sense the network using the SNMP protocol. Instead, it may launch a mobile agent, which updates all the MIBs of the network and returns to the administrator's station. Furthermore, it has been pointed out in [OK99] that if the number of nodes increases, the performance of the mobile agent approach improves, compared to the client/server solution. For the mobile agents, the performance curve obtained is almost flat with a smaller slope than the curve obtained for the client/server implementation.

In another work [RD99], a mobile agent solution is compared with a SNMP solution for the management of network elements. For a small number of network elements, the SNMP solution performs better than the mobile agent because it uses fewer and shorter messages to perform the task. As the number of network elements increases, the overhead associated with the SNMP solution turns the mobile agent into a more efficient approach.

As number of network elements increases, the response time for the mobile agents increases faster than the SNMP solution. In this case, mobile agents are larger and their migrations becomes more expensive. In short, it is shown that a mobile agent for managing network elements performs better than SNMP but only between certain bounds. Since the size of the agent is unbounded, the migration becomes more expensive as the number of network elements increases. Consequently, for a larger number of nodes, the SNMP solution outperforms the mobile agent solution.

This debate around mobile agents and message passing solutions has also been explored for information retrieval applications in general. In particular, the performance tests carried out in [BGM99] illustrate that inter-agent communication involves more overhead than RPC interactions and agent migrations involves even more overhead than inter-agent communication. Consequently, in the solution proposed by [BGM99], agents will make RPC-like calls across the network if they need to perform a few operations in a remote site. The agents only will migrate if a larger number of queries need to be performed at each site. The solution presented in [BGM99] shows a better performance for the mobile agent approach only when the number of queries increases.

The performance of a mobile agent solution depends not only on the behaviour of the agents but also on the platform where these agents are running. Mobile agent systems are still too complex to be used as low-level tools or building blocks for complex distributed applications. In [JS97], a performance analysis of the different functions that take part in the TACOMA mobile agent system is presented. The experiments give a median

execution time for the migration process which is four times slower than the corresponding SUN RPC solution for the same data.

A more detailed review of previous works is presented throughout the thesis when different parts constituting the proposed model are examined.

1.3 Contributions of The Thesis

This thesis has two major contributions. First, it proposes a model to solve a generic search problem based on software agent technology. The architecture involves two major approaches, mobile agents and stationary agents. It also comprises several components to guarantee the agent coordination and interactions as well as some fault tolerance mechanisms. For experimental purposes, the agents are implemented on a specific agent platform (TACOMA v1.2) but the same functionality can be achieved with any mobile agent platform without significant modifications.

In short, the thesis proposes to reduce a distributed search problem in a network of arbitrary topology to the implementation of a traversal algorithm applied to a logical spanning tree constructed on top of the original topology. Consequently, solutions for the single and multiple travelling agents are provided along with supporting components to implement agent coordination.

The second contribution of this thesis is the development of a prototype and the experimental results. There is an interesting debate among researchers on the mobile agents versus client/server or other message passing approaches. There is also a discussion about the performance of agents systems and agent based solutions. We develop a prototype, which explores several variables that may affect the performance of a mobile agent solution, such as topology, number of neighbours, size of the network and number of agents involved in the search. Consequently, the test results show that the number of agents has a positive direct impact on performance when the size of the network increases. However, increasing the number of neighbours per node does not improve the overall performance. Furthermore, the network topology does impact the overall performance. For these particular experiments, the binary spanning tree shows the best performance among other topologies tested.

1.4 Thesis Outline

The structure of the thesis is as follows. In Chapter 2 is described the agent model, as well as some available agent systems. Two non-commercial mobile agent systems are also surveyed in this chapter. The distributed search problem from an agent architecture perspective, along with solutions for the simple travelling agent case and the flooding scheme are discussed in Chapter 3. In Chapter 4 are examined several agent coordination models and their implementation using agent patterns. The applicability of those models to the distributed search problem is also addressed. The algorithms for the agent based distributed search as well as for the message passing solution are presented in Chapter 5.

Some experimental results are discussed in Chapter 6. Finally, in Chapter 7 are presented the conclusions, limitations and possible enhancements to the current model.

Chapter 2 Agents and Mobile Agent Systems

2.1 Introduction

On the Internet, an *agent*, also called an intelligent agent is a program that gathers information or performs some other service without the immediate presence of the user and is based on some regular schedule. Basically an agent, using parameters provided by the user, searches all or some part of the Internet, gathers information and presents it to the user in a daily or periodic basis. In [KT98], an agent is defined as a “program that acts on behalf of a user”.

According to [WJ95], there are certain properties that an agent should incorporate:

1. Autonomy: agents operate without the direct interventions of humans and have some kind of control over their actions and internal state.
2. Social ability: agents interact with other agents and possibly with humans.
3. Reactivity: agents perceive their environment and respond in a timely fashion to changes that occur within it.
4. Pro-activeness: agents do not simply act in response to their environment; they are able to exhibit goal-directed behaviour by taking the initiative.

A mobile agent is a program that can migrate autonomously, from node to node, to perform some computation on behalf of the user. Agents and particularly mobile agents

are very popular these days. The idea itself is not completely new. However, the combination of mobility, reasoning and knowledge representation gives agents an edge in today's computing industry. Agent computing saves bandwidth by moving computations to the resources they need. Agents carry the code to the remote resources and do not need the remote availability of a specific server. They can proceed without continuous network connections because interacting entities can be moved to the same site when connections are available and can then interact without requiring further network connections.

This last characteristic increases the asynchrony between client and servers. This feature is especially useful for mobile computers, which usually have unreliable low bandwidth connections. As agents execute concurrently, they also serve as a useful mechanism for parallel activities. An application can decompose its task in several agents to provide parallelism or fault tolerance.

There are two important concepts in the mobile agent technology: mobility and security. The mobility of an agent can be strong or weak. Strong mobility is found when data and code is transferred between two hosts. This is more associated with scripting capabilities rather than with simply transferring objects. On the other hand, weak mobility involves basically the transfer of data, usually: java code to a remote virtual machine and it does not allow mid-execution transfer. Another important element to consider is security. Malicious agents can be harmful to the host or to the network but also malicious host may be harmful to mobile agents. Usually, authentication using X509 certificates and encryption based on SSL are solutions to cope with these security threats.

Due to their mobile nature, agents might execute on heterogeneous machines and operating systems. Consequently, the portability of the agent code is a key issue. Most agents systems are based on interpreted programming languages such as Perl, Tcl and Python. However, scripting languages often have a lack of encapsulation and performance, which results in many agent systems choosing object-oriented languages, such as Java.

The technology still lacks secure and efficient execution support, standardization and coordination models. Coordination plays an important role in the development of mobile agents application. Some coordination techniques are described and applied later in this thesis.

A mobile agent system allows programs to be able to stop their execution, pack and migrate to a remote host. Once in the remote host they may resume the execution. The decisions on where to move and when are made by the agent autonomously. A typical mobile agent system is set up as a middleware residing on top of the operating system and providing support for the agent migration.

A mobile code platform can be described as *state-full* or *stateless* depending on how the migration process is addressed. In a state-full design, when an agent decides to migrate, the system captures the state information and transfers it along with the agent to the new host. Once in the new host, the agent resumes its execution. This process guarantees that the agent will resume its execution in a similar context, but now in a new host. Examples

of state-full mobile systems are Aglets¹, Voyager² and Carleton University Mobile Code Toolkit³.

An agent's state includes all its data, as well as the execution state of its thread. If this can be transmitted along with the agent, the thread can be reactivated on their destination site at the same point where the migration was initiated.

Unlike state-full environments, a stateless mobile environment does not capture any information about the running process. Any state information needed after migrating has to be saved explicitly by the agent before starting the migration procedure. Moreover, the execution is resumed at certain well-defined entry points that could be a predefined function. This approach frees the underlying system of saving and restoring the execution environment of an agent, a task that has been proven to be one of the most difficult. This simpler approach increases portability at the expense of transparency [JJ97]. Tacoma is an example of a stateless mobile code platform.

Current research on mobile agents are focussed on performance and planning problems. Agents need better mechanisms to determine when and where to travel. Usually, the decision about where to move is seen as an optimization problem. The factors that may affect the agent's decision are: location of services, network latencies, network bandwidth and machine loads.

¹ <http://www.tri.ibm.co.jp/aglets>

² <http://www.objectspace.com/voyager>

³ <http://gurjari.sce.carleton.ca/netmanage/mctoolkit>

Some planning algorithms to help agents make a better selection of their migration path through the network are discussed in [BGM99].

2.2 Mobile Agent Platforms

In this section, a brief description of the most known mobile agent platforms is presented. A more detailed survey of two non-commercial agent systems such as ARA, and TACOMA is presented in Section 2.3. The prototype developed herein, is based on these two systems, particularly on Tacoma. For this reason we will offer a closer look to them. The description is focussed on architecture, programming languages supported and implementation of the migration process.

2.2.1 Aglet

The Aglets Software Development Kit is an environment for programming mobile Internet agents in Java, developed by IBM Research. Its goal is to provide an easy and comprehensive model for programming mobile agents without requiring modifications to Java Virtual Machine or native code. Aglet is a state-full mobile system. When the aglet moves, it takes along its program code as well as the states of all the objects it is carrying. The Aglet abstract class defines the fundamental methods, for example, *dispatch(URL)*, which is used to control the mobility and life cycles of mobile agents. All mobile agents defined in Aglet have to extend this abstract class. The dispatch method causes an aglet to move from the local machine to the destination specified as its argument.

2.2.2 D'Agents

The mobile-agent system called D'Agents is under development at Dartmouth College. The ultimate goal of D'Agents is to support applications that require the retrieval, organization and presentation of distributed information in arbitrary networks. Some of the research areas are: Security mechanisms, support for mobile and partially connected computers, navigation, network sensing and resource discovery tools, automatic indexing, retrieval and clustering techniques for text and other documents. It supports agents written in Tcl and Java as well as stationary agents written in C/C++.

2.2.3 Concordia

Concordia is Java-based mobile agent system developed by Mitsubishi Electric with a strong focus on security and reliability. The movement of Concordia agents is based on an itinerary of places to visit. During the migration the agent object code as well as the data is moved from one machine to another, but not the thread state. Agents can be saved to a persistent store before leaving a site or after arriving in order to avoid agent loss in the event of a server crash. Concordia implements a weak mobility mechanism requiring that agents be serialized [BGM99]. Security in the hosts is guaranteed by using cryptographic authentication of the agent's owner as well as access control list that guard each resource.

2.3 Ara: Agents for Remote Access

The Ara system is a mobile agent platform developed at the University of Kaiserslautern, Germany. A mobile agent in Ara is a program able to move at its own choice. These agents can be programmed in several programming languages. The platform also provides facilities for access to system resources as well as mechanisms for agent communication.

There are two important components in the Ara system: Agents and places. The programming model consists of agents moving between and staying at places. When an agent is on a place, it may use certain services provided by the host system or by other agents. A place is physically located on some host computer and may impose specific security restrictions to the agents staying at the place.

Two important aspects in the design of mobile systems are portability and security. Usually, existing mobile agent platforms do not run their agent in real machine processors and operating systems. Instead, they restrict the actions of the agent to a more closed environment controlled by an interpreter and a run-time system. In Ara, mobile agents are programmed in some interpreted language and executed within an interpreter for this language using a special run-time system for agents, called Core in Ara terms [PS97].

Ara provides an interface to attach existing programming languages. This characteristic makes it possible to employ several interpreters for different programming languages at

the same time, on top of the core, which provides the services to the agents. The process of adding a new interpreter is well defined in the Ara documentation. Currently, interpreters for Tcl and C/C++ have been adapted to the Ara core. The Figure 2.3.1 shows the architecture of the Ara system.

Figure 2.3.1 The Ara system architecture [PS97]. Example for languages A and B

Agents in the Ara systems are executed as independent threads. Processes are also used for certain internal tasks and are also called system processes. The use of threads guarantees that the management of agents is kept under the control of the core as well as a benefit in performance.

The Ara platform falls into the category of the state-full mobile code systems. Mobile agents in Ara carry along their execution environment. Agents can migrate at any point of their execution by simply calling the *ara_go* function. The migration function moves the entire agent to a new place and resumes in the exact state where it left off. More specifically, the first instruction is to execute right after the *ara_go* function call. The process of migration does not affect either the flow of execution or the agent's internal

data. The programmer is unaware of any work regarding the capture of the state, transmission and restoration of the agent in the new place.

In terms of agent communication, Ara emphasizes local agent communication by choosing client/server style interaction based on message exchange. For this purpose, the core provides a concept of service points. Service points are meeting points with a well-known name where agents located at a specific place can interact as client and servers through an exchange of asynchronous messages.

Security in the Ara architecture is guaranteed by using the interpreters for basic memory protection and by imposing restrictions at the places. A place decides on the conditions of admission of an agent applying to enter. These conditions specify access rights to various system resources such as files, CPU, memory or disk space. On the other hand, fault tolerance is achieved by providing basic means of recovery from accidents such as, host crashes or line breakdowns. An agent can create a checkpoint (complete record of its internal state) at any time during its execution [PS97]. Checkpoints are stored in some persistent media, usually on disk, and can be used later to restore the agent. Applications may build their own fault tolerance schemes on top of this mechanism, however the system also provides a mechanism to automatically checkpoint all local agents in case of an emergency shutdown.

Current works on the Ara platforms focus on the security implementation. The agent will be able to create places with programmable admissions policies. The control of local

resources, currently limited to CPU time and memory, will be enhanced to regulate files, network connections, disk space, bandwidth and visited places [PS97]. In addition, tools for monitoring and visualizing a set of Ara systems distributed across the network are also under development.

2.4 TACOMA

The TACOMA project is collaboration between the Department of Computer Science, University of Tromsø, Norway, the Department of Computer Science, Cornell University, Ithaca, New York, USA and the Department of Computer Science, University of California, San Diego, USA. Its main objective is to provide operating system support for agents to solve problems traditionally addressed by other distributed computing paradigms, e.g. the client/server model. TACOMA is implemented in C and its version 2.0 for UNIX supports several programming languages such as (C/C++, Tcl/Tk, Perl, Python)

The key element in the TACOMA architecture is called "briefcase". This encapsulating structure contains a set of substructures called folders. Folders contain data and code needed by the agent. Furthermore, we can say that the agent itself is a briefcase that is delivered between hosts. TACOMA also provides some persistency mechanism, allowing an agent to save folders, which acquire the name of cabinets and can be left on hosts that the agent has visited.

Each folder has an ASCII name. There are well-known folders that guarantee a uniform behaviour or at least establish certain rules that govern the process of constructing an agent in TACOMA. For instance, a folder called CODE contains the code to execute on every host. Also, a folder called HOST defines the list of hosts that the agent has to visit during its journey. Another folder named RETURNHOST specifies a host where the agent has to return when the HOST folder is exhausted or in case of error.

Figure 2.4.1. Briefcases and Cabinets

By defining the HOST folder properly and modifying it, an agent can modify its itinerary depending on pre-established criteria. Examples of traversal algorithms, which make use of this characteristic, will be shown in the next sections of the thesis.

Architecturally, TACOMA is composed of two major elements: a firewall and systems agents. The firewall is responsible for accepting and delivering briefcases, while the system agents are stationary agents, which complement the job of the firewall. These agents, for instance, are responsible for executing the code, which could be C/C++, Perl,

etc. All the functionality in the TACOMA system is provided via an API. The API allows operations on folders, cabinets and provides the main function for initiating the migration process, the *meet* function.

Migration is the hallmark of mobile computing. In TACOMA, the function *meet* starts the migration process. This action ensures that the entire briefcase is delivered to the first host in the HOST folder. The *meet* operation specifies an agent, which resides in the remote host and which receives the briefcase. Usually, a *meet* operation is performed by specifying a system agent (i.e. *ag_c*), which in fact, is responsible for executing the C/C++ code contained in the CODE folder. The following diagram depicts the migration process and the relationship between the different components in the TACOMA system.

Figure 2.4.2 Migration process in Tacoma

According to the Figure 2.4.2 the agent running on the host 1 initiates the migration process by calling the *meet* function. The entire briefcase is sent to the host 2. The agent on host 1 continues running and the new agent starts its execution on host 2 from a well-defined point, by default a function called *Main* (notice the capital letter).

In terms of performance, TACOMA focuses on briefcase delivery and restoration of the execution environment on the remote host. In particular, when using agents implemented in C/C++, the folder containing the agent's code is saved into temporary files, compiled and linked with predefined stubs. This process adds some delay inherent to the invocation of the C/C++ compiler. Other mobile code platforms like Ara [PS97] move byte code generated by a cross compiler. In particular, Ara uses a C/C++ compiler such as gcc configured as a cross compiler for Motorola 680X0 target architecture. After receiving the agent, an interpreter (i.e. MACE) is called to interpret the byte code.

TACOMA does not rely on any specific language to guarantee the host integrity. It uses the underlying operating system mechanism to accomplish it. This characteristic increases flexibility but can also impact performance, depending on the operating system. A more detailed performance analysis can be found in [JS97].

The current development of TACOMA is focused on fault-tolerance, security, applicability and management issues. The TACOMA platform has also been ported to new operating system architectures, in particular Windows NT, Windows CE and the PalmOS.

There exists a modification of TACOMA for Personal Digital Assistant (PDA) environments. In mobile computers, resources such as RAM, CPUs and operating system support for inter process communication, are scarce or are not as developed as in

workstations. Therefore, the TACOMA version for such environments, so-called TACOMA Lite, intends to improve memory management, and minimize memory consumption by reducing stack size and local variables. Moreover, the API is simplified to cope with the constraints posed by a mobile environment.

Several applications have been developed for PDA environments. “Weather Alarm” is an example that makes use of TACOMA Lite to create an agent, which can be programmed to monitor weather parameters, like wind speed, radiation levels and alerts the user if a threshold value is exceeded. Also, other applications like “Stock Ticker” and “Mobile Diary” have been developed [JJ97].

Chapter 3 Distributed Search and Mobile Agent Computing

3.1 Introduction

In a highly interconnected environment like Internet/Intranet, any computer can be reached from any other computer by establishing a peer-to-peer communication. In this kind of environment, one may assume that there exists a direct linkage between two arbitrary computers on the network. Consequently, given any set of computers of size N , the subnet formed by those processors can be considered as a complete network.

In an environment of cooperating processors, such as the computational grid presented in [CF99], it is common to find an extensive use of remote resources (files in remote file systems, URLs, etc). Frequently, the physical location of remote resources is unknown. Consequently, applications need a mechanism to efficiently locate resources in a transparent manner. The transparency of remote computational resources can be achieved by applying distributed search mechanisms.

In order to simplify the distributed search problem in a complete network environment, we propose the construction of a logical spanning tree on top of the physical topology. By creating this logical spanning tree, the search problem can be reduced to a traversal problem in a spanning tree.

In this thesis, we focus on the distributed search problem from a mobile agent perspective without any emphasis on the spanning tree construction. It is assumed that the spanning tree is constructed in a pre-processing phase and updated every time a new node is registered in the network. Figure 3.1.1 shows a logical mapping for a network of $N=8$ processors.

Figure 3.1.1 Logical Spanning Tree Mapping.

Each node will contain information about its immediate neighbours, and this information will be used to decide where to travel and in which order. The first and simplest way of traversing this tree is to have a single agent travel across the network. We will study the impact of the topology on the computational complexity in terms of time and number of messages. Initially, we will focus on the search algorithm rather than the search criteria. It is assumed that the agent interacts with the local resources at each node and carries along certain information that satisfies some predefined criteria, e.g. all file names that match certain pattern or simply the content of a predefined file.

3.2 The Single Travelling Agent

The problem of the agent based distributed search has been addressed in several published research articles. In particular, the travelling agent problem (TAP) has been analysed as an analogy of the classic travelling salesman problem (TSP). In [MC98] the travelling agent problem is described as the minimal sequence of jumps that an agent has to accomplish in order to minimize the expected time needed to complete a given task. If all sites must be visited, the problem is then reduced to the classic TSP, which is known to be NP-Complete.

Formally, the TAP is defined as follows [MC98]:

There are N sites, S_i $0 < i \leq N$. Each site has a known probability $0 \leq p_i \leq 1$ of being able to complete the agent's task at a time $t_i > 0$. The probabilities are independent, and the travel time for an agent to move from S_i to S_j is also known and is given by $L_{ij} \geq 0$. The problem of the travelling agent is to minimize the expected time to complete the task.

Assuming that the travel times L_{ij} 's are constant and equal to L , with arbitrary t_i and p_i , it is proved in [MC98] that the TAP can be solved in polynomial time if the nodes are visited in a decreasing order of $p_i/(t_i+L)$.

In this thesis, we apply the idea of [MC98] to our logical spanning tree representation in order to implement the traversal algorithms for the case of a single travelling agent. For simplicity, we will assume that the probability p_i to complete the task at any given node S_i

is constant and equals to P . With this assumption, the ordering will depend basically on the time t_i that is required to accomplish the task at the site S_i .

Initially, we will focus on describing the traversal algorithm and its implementation using TACOMA agent system. The ordering is applied to the local neighbour information and will be described in more detail in Chapter 4 when we examine coordination models and the agent router in particular.

The traversal algorithms implemented on TACOMA make an extensive use of the HOST folder as a mechanism to control the migration process and define the itinerary of the travelling agent. To traverse the spanning tree, the agent may start from any host in the network, reading its local neighbour information and travelling to the first processor on the list. Every time it arrives at a new node it reads the neighbour information and modifies its HOST folder accordingly. Consequently, the mechanism chosen to insert new hosts in the HOST folder would result in different tree traversal algorithms such as DEPTH_FIRST, BREADTH_FIRST or any particular ordering as discussed previously.

The criterion for terminating the search could be either to find the first K matches for the search or to visit all the sites of the tree. The former could be addressed by using an incomplete search algorithm like “branch and bound”. Incomplete search algorithms allow pruning the search space considerably by selecting upper and lower bounds and disregarding paths outside those boundaries. For the time being, we assume that all nodes have to be visited. Thus, a complete search algorithm will be implemented.

The Figure 3.2.1 shows how the HOST folder is modified in order to implement two kinds of complete search techniques. The figure also shows how to guarantee that the agent will return to the original host when all nodes have been visited.

Figure 3.2.1. Modifying the HOST folder to implement traversal algorithms.

Note that when the HOST folder ends with an empty host name, it means that the agent has to return to the host specified in the RETURNHOST folder. After returning to the initiator, the agent contains the path of all hosts visited in its HOST folder.

In order to avoid backtracking in these two algorithms, the agent stores the information of the neighbours that have to be visited. Therefore, the size of the HOST folder as well as the size of the agent grows as it travels along the tree. This particular characteristic can be regarded as a drawback compared to the message passing implementation. The advantage of using mobile agents could be in the logic of the application. An intensive search or multiple queries may then justify the applicability of a mobile agent rather than an intensive exchange of messages.

3.3 Flooding the Network

Another approach to solve the traversal problem is to apply a flooding technique. In a flooding scheme, each entity of the network, after receiving a message, propagates the message to all of its immediate neighbours except the node from which the message arrived. Flooding is a way of distributed routing information updates quickly to every node in a large network and it is often used in multicast packets (from one source to many specific nodes in a real or virtual network).

Similarly, in a mobile agent environment, the agent starts from a certain node of the network and travels simultaneously to every host in its neighbour list. In other words, it duplicates itself by generating as many identical agents as hosts specified in the local neighbour list. However, the HOST folder will contain only the host to which the agent is travelling. Therefore, the size of this folder is bounded and independent of N (size of the network).

This feature represents an advantage over the single mobile agent approach. Moreover, we can assume that the size of the agent as well as the amount of the data gathered during the journey is bounded. Consequently, we can state that the entire size of the briefcase is bounded and independent of the size of the network.

Since the agents duplicate themselves every time they arrive at a new node, the number of active agents at a given time is unknown. This number depends on the topology chosen. For example, when an agent reaches a leaf node in a tree topology, the agent

returns to the initiator with the information gathered. The initiator will receive as many agents as leaf nodes defined for the tree. Unfortunately, the number of leafs is unknown to the initiator and an extra mechanism has to be implemented to address termination detection.

In order to control the creation of new agents, the initiator needs a mechanism to register the birth and arrival of agents. With this mechanism, the initiator is able to solve the problem of termination detection. This new component associated to each node is called *Blackboard*. The blackboard allows agents to register every time they duplicate themselves. The blackboard, if requested, will inform the number of agents registered at a given time. There exists one blackboard per node and agents will report to the initiator's blackboard. All agents know who the initiator is because this information travels in the RETUERNHOST folder.

The blackboard supports the following operations:

1. Add an agent ID (Register a new agent),
2. Delete an agent ID,
3. List the agents registered in the blackboard,
4. Count the number of agent currently registered in the blackboard.

The *delete* function returns the number of agents that remain in the blackboard. It consists of a sequence of *delete* and *count* commands. All these functions are atomic in order to avoid any race condition or deadlock. In particular, by using the *delete* function, an agent

may detect whether it is the last agent that has arrived at the initiator (remaining elements = 0), which means that the search has been completed.

Along with the blackboard and TACOMA's systems agents, every node of the network is also provided with stationary agents that guarantee the coordination among the travelling agents. The responsibilities of those agents (whiteboards, routers, etc) and their interactions are addressed in the next chapter on agent coordination.

The Figure 3.3.1 shows the migration process for the flooding algorithm and the different components that take part in the process.

Figure 3.3.1 Cloning of Agents.

In part A, the agent A will clone itself, generating k agents $A_1, \dots, A_k$ (k: number of neighbours). Each newly generated agent will report to the blackboard before it migrates. When the migration is performed from a leaf node to the initiator (part B), the arriving

agent reports to the blackboard. When all agents have travelled to the initiator node, a condition that is verified by querying the blackboard, the algorithm terminates.

Chapter 4 Coordination Models in Distributed Search

4.1 Introduction

When decomposing a task into several independent sub-tasks, agents need strategies to coordinate their actions. Coordination is very important to avoid duplication of efforts or simply to detect special cases, such as termination detection. Coordination models may vary and their use depends on the requirement of the application.

This chapter examines several coordination models that can be applied to an agent based distributed search. The most distinctive characteristics of each model are presented along with a comparison among the different models. Several articles are surveyed to support the discussion on coordination models and other related issues. The last sections present the coordination models selected for the implementation of the agent based distributed search.

4.2 Various Coordination Models

When agents need to interact with each other, they can establish a peer-to-peer connection to exchange data and synchronize actions. This method is called “direct coordination”. Agents may also agree to rendezvous at certain places to interact with

other agents. This method is called “meeting oriented coordination”. In addition, agents can make use of persistent structures to store data on the sites they visit. The information left on the sites can be read later by incoming agents. This kind of coordination is known as “whiteboard based coordination”.

4.2.1 Direct Coordination Model

In direct coordination models, agents start a communication by explicitly naming the partners involved. To establish communication, they have to agree on a communication protocol. Several agent systems adopt the direct coordination models allowing agents to communicate based on object-oriented paradigms such as remote object invocation or simply TCP/IP. This approach might produce an increase in network traffic and requires stable network connections. Several studies on coordination models ([CL00], [GB99]) recommend the usage of direct coordination for agent to local environment interactions by providing local servers at each site.

The blackboard component introduced in the previous chapter is an example of direct coordination model applied to the distributed search. In this particular case, each agent opens a TCP/IP connection with the initiator’s blackboard to inform about the creation of a new agent. This interaction involves two messages, one to inform the identifier of the new agent and the other for the corresponding acknowledgement. When an agent returns to the initiator, it performs a local interaction with the blackboard to notify its arrival.

4.2.2 Meeting Oriented Coordination Model

The meeting oriented coordination technique can also be implemented locally or between remote sites. Usually, agents that take part in a meeting know certain meeting points where they can start a meeting operation. In the Ara mobile agent system [PS97], the meeting operations are implemented as synchronization objects. Agents share references to these objects and are able to join a meeting by accessing one of these shared structures. When deployed among several sites, the meeting oriented coordination relies on message passing techniques to implement the interactions. In this case, meeting can be seen as a particular case of direct coordination.

Previous research on agent based distributed search have experimented the use of local meetings to coordinate agents. When an agent has concluded its task at a given site, it creates a meeting agent, which is left at that site. Afterwards, when new agents arrive, they can communicate with the meeting agent, which will inform that the site has already been visited. This approach produces less network traffic while forcing agents to remain at each site for an indefinite time.

4.2.3 Whiteboards, Tuples and Linda-like Coordination Models

Whiteboards, Tuples and Linda-like coordination models have a lot in common because they intent to uncouple the participating entities in time and space. Due to the similarities of these three models, they are discussed together in this section.

In the whiteboard coordination model, agents interact via shared data structures. These structures are used as common repositories to store and retrieve messages. Usually, whiteboards are associated with the local environment. Agents can leave messages to be read later by another agents. This approach allows the local hosting environment to control the access to the shared data leading to a more secure execution environment than in the previous models.

Linda-like coordination [CL00] is a special kind of whiteboard coordination used to control, communicate and synchronize mobile agents. Linda provides an abstraction for programming concurrent agents and defines a small set of coordination options. In a Linda based system, a group of agents work on a task within a shared environment called tuple-space. Tuples are structured containers, which contain information relevant to the application. Linda provides two blocking functions to retrieve data from the tuple space (*in* and *rd*). Consequently, agents block until a matching tuple becomes available.

One of the advantages of Linda-like coordination over direct coordination is that agents do not need to know each other to communicate. Tuple based coordination uncouples coordinating agents in space and time. Linda-like coordination is available for a number of programming languages and hardware architectures and is also used in several applications based on Web technologies [CT98][S95].

In [CT98] is presented a reference architecture to support agent-based applications on the Web. The architecture combines the Web technology for access and presentation, Java as

the execution mechanism and Linda-like coordination to manage the interactions among agents. The application can be seen as a combination of several independent agents: interface agents, home agents, gateway agents, application agents and a shared workspace where the agents live and communicate. The coordination mechanism is implemented in Jada, which is a language that extends Java with the Linda coordination primitives. Jada is based on a set of classes to be used to access shared object spaces. Following the Linda coordination model, Jada defines a method *out* to put an object into the shared workspace and *read* to get an object from the object space. It is concluded in [CT98] that combining Linda-like coordination with Java makes it possible to build a flexible platform to support distributed applications easily and without extensions to the building blocks used.

Several mobile agent coordination models for Internet applications are examined in [CLZ00] through a case study application. The authors assume that HTML pages, spread over the Internet, contain information and a given user needs to find all the pages that contain a given keyword. The search agent then travels to a site, analyses the documents and follows the links by cloning itself. This approach is compared with a more classical client/server approach in which the client application retrieves an HTML document from a Web server, analyses it and repeats the same process for all the links in those documents that contain the keyword.

In the particular case of the HTML search agent, it is possible that the agent follows a link to a document containing a cross-reference, causing the agent to go back and thus visiting the same host more than once. In order to avoid this situation and improve the

agent-based approach, several coordination models and their pros and cons based on the HTML search agents are discussed in [CL00].

The authors propose a reactive tuple model to implement agent coordination. Reactivity provides an extra level of intelligence by associating some actions to the messages left on the sites. In the mobile agent solution for the HTML search, reactive tuples are left on each site to avoid multiple visits to the same site and to mirror sites. The tuples left on the sites contain a reactive code, which guarantees that every time a tuple is created, a copy of the tuple will be also created on the mirror sites. Consequently, this replication system guarantees that multiple visits are avoided not only to the same site but also to mirror sites.

4.3 Initiation of Agent Interactions

An important aspect to consider when designing a mobile agent application is how to initiate the agent interactions. This may turn particularly difficult in a highly mobile environment. Many of the current static agent-based applications intended to help users to make a more efficient use of Internet are based on the Client/Server model. This model requires direct connection and do not take advantage of software mobility. In [FK97] is presented a generic structure based on Abstract Syntax Notation (ASN) to facilitate agent portability between different applications. This representation provides a generic structure, which intends to increase flexibility to agent based multimedia applications.

In the model proposed by [FK97] agents can query the remote site before migrating. By doing so, agents can find out about particular aspects of the remote hosts such as resources, languages supported, availability, etc. This mechanism helps agents to avoid unnecessary migrations but might increase the number of messages or interactions between hosts if employed every time an agent intends to migrate.

An agent-tracking scheme is proposed in [FK97] as a mechanism to initiate the interactions with remotely located agents. In particular, the initiator or launching application may know about the travelling agent's itinerary and may inquire at each site in order to determine its whereabouts. This solution can be very expensive and is perhaps ineffective in a highly mobile environment where the agent requiring contact moves continuously from host to host. Another alternative, also recommended by [FK97], proposes that the moving agent send a lightweight message every time it arrives at a new host. Assuming that the initiator needs to contact the agent, perhaps to obtain certain partial information or simply to stop its execution, this approach seems to incur the same problem if the initiator replies and the agent has already left the site.

To cope with this situation and assuming that the agent itinerary is known, we recommend the alternative of leaving a notification or message at certain host (not yet visited) indicating that the initiator needs to contact the agent. By applying this variant, it is the travelling agent, which initiates the direct communication upon arriving to a marked site (site containing a message). A mechanism to efficiently post those messages would be an interesting topic to investigate.

4.4 Proposed Coordination Model for Distributed Search

It is not known whether any coordination model is most suitable for mobile agent applications. The usage of one model over others depends on the application and performance reasons may suggest the use of direct coordination rather than migrating the entire agent to interact with local reactive tuples or messages.

4.4.1 Whiteboard Coordination

In the flooding algorithm, more than one agent may visit the same site. This situation occurs when the algorithm is started from a node other than the root of the logical spanning tree. This is the most likely situation and the algorithm needs a mechanism to control and avoid multiple visits to the same site. The flooding algorithm presented in this thesis introduces the whiteboard-based coordination in order to cope with multiple visits to the same site.

The implementation of the whiteboard for distributed search is based on the whiteboard pattern as depicted in Figure 4.4.1.1. This pattern solves the problem of how agents can exchange location specific data with other mobile agents. Usually an agent wants to store data at a place so that other agents can read it. The solution presented by this pattern provides an area at every place where agents can leave messages for each other.

Figure 4.4.1 The Whiteboard Pattern [DW99].

When an agent arrives at a node, it first examines whether the site has been visited previously. If so, it just returns home. Due to the hierarchical structure of the network, when a node has been visited, it makes sense to assume that the neighbours below this node have also been visited or at least that there are agents travelling towards them. Consequently, the incoming agent realizes that the cloned version of it has been there before and that it has to return to the initiator and report its partial search results.

The implementation of the whiteboards takes advantage of the persistent mechanism provided in TACOMA, called file cabinets. This structure allows an agent to leave information in the form of TACOMA's folders to be read by incoming agents.

4.4.2 The Agent Router

In general, a router can be defined as device or software in a computer that determines the next network point to which a packet should be forwarded. Usually, a router may create or maintain a table of the available routes and their conditions and uses this information along with distances and cost algorithms to determine the best route for a given packet. Analogously, in a mobile agent environment, a router can be defined as a stationary agent that determines the next host to which a certain mobile agent should travel. This routing agent may also maintain tables with the information needed to decide the most satisfactory route or itinerary.

In our scheme, mobile agents request information about the local neighbours when they arrive to a site. This information is provided via a stationary agent called an agent router. This interaction can be seen as a particular case of meeting interaction. Every node of the network has a stationary agent router, which provides information about local neighbours but also may provide information about neighbours of any other node of the network.

This stationary agent acts as a local or global router to facilitate the navigation of mobile agents. For example, let's think of a single travelling agent, which has arrived to the host A and meets with the local agent router to find out about the immediate neighbours. The agent router provides the list of neighbours $N_1, \dots, N_k$. Following any of the traversal algorithms for the single travel agent, the agent will attempt to travel to the node N_1 and insert the remaining neighbours into its list of pending hosts. Unfortunately, N_1 is down or the agent could not complete the trip. Then, the agent will contact the agent router now

as a global router to find out the neighbours of N_i . By doing so, the agent guarantees that only the faulting node will be skipped and that the sub-tree below will be explored.

By applying this simple technique, either a single travelling agent or a flooding algorithm can cope with the problem of temporary failure of nodes without compromising the entire network. Note that at a certain time and with several nodes down, the logical structure is affected and the network can be logically disconnected. Agent routers, configured as global routers and spread across the network, will then guarantee a functional behaviour as well as a graceful recovery when faulting nodes become active again.

The implementation of the agent router is based on another agent pattern, the router pattern (Figure 4.4.2). The objective of this pattern is to solve how agents can select a destination where a task can be performed best. To solve this problem, the router bases its selection on profiles associated with the destination sites. These profiles depend on the logic of the application. In our particular case, we propose to select the profiles according to the result presented in Chapter 3. Consequently, the agent router will sort the list of local neighbours based on an estimation of the time required to perform the task at each node.

Figure 4.4.2 The Router Pattern [DW99].

The agent router supports the following set of operations:

1. Set local neighbour list.
2. Get local neighbour list.
3. Define global neighbour list.
4. Get global neighbour list.

Options 3 and 4 allow us to get/set the local neighbour list of any node in the network if agent router is configured as a global router.

Unfortunately, the agent router is not enough to guarantee a robust behaviour of the system. More sophisticated mechanisms have to be put into place to cope with the failure of the host and agent itself. Some systems propose to copy the agent to a persistent location before departing and after arriving to a new host. Others implement the so-called checkpoints where the agent platform saves the agent and its state periodically to enable it to restore the environment in case of a server crash.

Currently in our implementation, the initiator is unaware of any failure of its cloned travelling agents. A timeout mechanism is now implemented to notify the initiator that the agent has not arrived after a reasonable amount of time. This “reasonable” timeout depends on the logic of the application and it is very debatable. Furthermore, there are some unanswered questions such as, how global routers will be assigned, when to update the local neighbour information and how to replicate changes to the global routing tables. Reactive tuples may be an answer to the problem of replicating the changes, as in [CLZ00]: changes to a global router can be propagated to the other global routers in the same way that messages posted into a Web site are propagated to its mirror sites. Future enhancements to this work might focus on these issues.

Chapter 5 Agent and Message Based Solutions

5.1 Introduction

Previous chapters have introduced and explained the different components of the proposed mobile agent architecture. This chapter presents the distributed search algorithms from the message passing and mobile agent perspectives. In both approaches, the single travelling agent/message and the flooding technique are shown separately.

Both solutions, messages and mobile agents, consist of two major parts: the launching application and the travelling message or agent. The launcher application is responsible for packaging the agent into a TACOMA briefcase and injects it into the network. Once the agent or message has been sent out, the launcher or initiator blocks until the search is completed.

The agents are implemented as a very simple finite state machine, with three states: INIT, SEARCHING and DONE. Initially, when the initiator creates the agent or agents, their state is set to INIT. When the agents reach the first searchable node in their itinerary, their state is changed to SEARCHING. As the agent or message moves across the network, it remains in the SEARCHING state. Only when the HOST folder is exhausted, or in other words, when the agent has fulfilled its itinerary, the state is changed to DONE.

Agents or messages in DONE state have to travel to the initiator to report their partial search results.

The interactions between agents and the stationary components responsible for the coordination are represented in the algorithms by employing supporting functions. In particular, the interactions between the mobile agents and the agent router are encapsulated in the functions *Get_Local_Neighbours* and *Get_Global_Neighbours*. The *Get_Local_Neighbours* function performs a local meeting with the agent router in order to obtain the list of immediate neighbours. On the other hand, the function *Get_Global_Neighbours* is able to contact either a local or remote agent router and make inquiries about immediate neighbours of any node in the network.

The direct coordination mechanism can be found in the backboard related functions: *Insert_AgentID_in_Blackboard* and *Remove_AgentID_from_Blackboard*. Both functions establish a TCP/IP connection with the initiator's blackboard in order to set and retrieve agent related information. Similarly, the whiteboard-based coordination is implemented by the functions *Get_AgentID_From_Whiteboard* and *Set_AgentID_In_Whiteboard*.

5.2 The Agent Based Algorithms

This section presents the algorithms for the single travelling agent and the flooding scheme. These algorithms describe the behaviour of the mobile agents after being injected into the network by the initiator or launching application. As it was mentioned in

the introduction, the agents are implemented as a finite state machine. When the algorithms start the mobile agent is in state INIT. This means that the agent was successfully created and it is ready to interact with the local resources of the first searchable node in its itinerary.

Every time the agent arrives at a node other than the initiator, it performs a local search and carries the result in a special folder called *DATA*. If the agent reaches the initiator and its state is set to DONE. The agent then contacts the launching application to inform the partial search results.

After performing a local search, the agent retrieves the list of local neighbours. To achieve this, the local agent router is contacted by calling the function *Get_Local_Neighbours*. The list of neighbours obtained is inserted in the HOST folder. This is the folder that the agent carries along and specifies the itinerary. Different implementation of the *Insert_Local_Neighbours* function may produce different itineraries. Furthermore, some artificial intelligence techniques can be incorporated to this function to enhance the "agent's intelligence".

If the list of neighbours returned by the local router is empty, it means that the node does not contain any immediate neighbours. In this case, the agent sets its state to DONE and returns to the initiator. If the list of neighbours is not empty, the agent attempts to travel to the first node in the list. If the agent cannot complete the migration because the remote node is temporarily down or due to any communication error, the agent contacts the

global router. When a global router is contacted, it informs the list of immediate neighbours of that faulting node. By applying this mechanism, the algorithms guarantees that only faulting nodes are not visited and the sub tree below that node will be explored.

5.2.1 The Single Travelling Agent

```

If INITATOR and STATE=DONE
  Begin
    Inform_Partial_Results
    Exit;
  End
Else
  STATE=SEARCHING
  DATA = Perform_Local_Search
  Get_Local_Neighbours (ROUTER)
  Insert_Local_Neighbours (HOST_LIST)
  For each Ni in HOST_LIST
    Begin
      If Ni==NULL // Ni is the last host in list
        Begin
          STATE=DONE
          Travel to INITATOR
          Exit
        End
      Travel to Ni
      If cannot_travel to Ni
        Begin
          Get_Global_Neighbours (Ni, ROUTER)
          Insert_Local_Neighbours(HOST_LIST)
        End
      Else Exit
    End
  End

```

Note: Appendix A (Figure A.1) shows the flow chart for the single travelling agent.

5.2.2 Flooding Algorithm for Mobile Agents

```

If INITIATOR and STATE=DONE
  Begin
    Inform_Partial_Results
    Remove_AgentID_from_Blackboard
    If Blackboard_Empty
      Notify_Launcher_Application
    Exit;
  End
Else
  STATE=SEARCHING
  Get_AgentID_From_WhiteBoard
  IF AgentID // this host has been already visited
    Travel to INITIATOR
  Else
    Set_AgentID_In_Whiteboard
    Perform_Local_Search
    Get_Local_Neighbours (ROUTER)
    Insert_Local_Neighbours (HOST_LIST)
    For each  $N_i$  in HOST_LIST
      Begin
        If  $N_i$ =NULL //  $N_i$  is the last host in list
          Begin
            STATE=DONE
            Travel to INITIATOR
            Exit
          End
        Travel to  $N_i$ 
        If cannot_travel to  $N_i$ 
          Begin
            Get_Global_Neighbours ( $N_i$ , ROUTER)
            Insert_Local_Neighbours (HOSTLIST)
          End
        Else
          Insert_AgentID_in_Blackboard
        End
      End
    End
  End
End

```

Note: Appendix A (Figure A.2) shows the flow chart for the flooding scheme.

5.3 The Message Passing Algorithms

Similar to the mobile agent approach, the message passing solutions utilize the TACOMA system as a communication platform. In this particular case, the agents are stationary. The communication is limited to the exchange of a briefcase, which contains data and additional information needed to coordinate agent actions, such as the global router and the initiator. The stationary agents will exchange messages according to the same techniques employed for the mobile agent case. Consequently, this allows us to compare the solutions within the same group (agents or messages) or between the two approaches.

The following is a description of the algorithms for the single message and the flooding scheme. As in the previous algorithms the stationary agents are implemented as very simple finite state machines with three states: INIT, SEARCHING, and DONE. The initiator or launching application will now create a briefcase with the essential folders: HOST, ROUTER, DATA, STATE, etc and it will send the briefcase to the first node in the HOST list. From that moment onwards, the algorithms behave similar to mobile agents but now stationary agents will receive the briefcase and will append the local search results rather than creating and activating new agents dynamically.

5.3.1 The Single Message Algorithm

```

If INITIATOR and STATE=DONE
  Begin
    Inform_Partial_Results
    Exit;
  End
Else
  STATE=SEARCHING
  Perform_Local_Search
  Get_Local_Neighbours (ROUTER)
  Insert_Local_Neighbours (HOST_LIST)
  For each  $N_i$  in HOST_LIST
    Begin
      If  $N_i \neq \text{NULL}$  //  $N_i$  is the last host in list
        Begin
          STATE=DONE
          Send Message to INITIATOR
          Exit
        End
      Travel to  $N_i$ 
      If cannot_send_message to  $N_i$ 
        Begin
          Get_Global_Neighbours ( $N_i$ , ROUTER)
          Insert_Local_Neighbours (HOST_LIST)
        End
      Else Exit
    End
  End
End

```

Note: Appendix A (Figure A.3) shows the flow chart for the single message.

5.3.2 The Flooding Algorithm For Stationary Agents

```

If INITIATOR and STATE=DONE
  Begin
    Inform_Partial_Results
    Remove_MessageID_from_Blackboard
    If Blackboard_Empty
      Notify_Launcher_Application
    Exit;
  End

```

```

    End
Else
    STATE=SEARCHING
    Get_MessageID_from_WhiteBoard
    IF AgentID // this host has been already visited
        Send Message to INITIATOR
    Else
        Set_MessageID_in_Whiteboard
        Perform_Local_Search
        Get_Local_Neighbours (ROUTER)
        Insert_Local_Neighbours (HOST_LIST)
        For each  $N_i$  in HOST_LIST
            Begin
                If  $N_i$ =NULL //  $N_i$  is the last host in list
                    Begin
                        STATE=DONE
                        Send Message to INITIATOR
                        Exit
                    End
                Travel to  $N_i$ 
                If cannot_send_message to  $N_i$ 
                    Begin
                        Get_Global_Neighbours ( $N_i$ , ROUTER)
                        Insert_Local_Neighbours (HOSTLIST)
                    End
                Else
                    Insert_MessageID_in_Blackboard
                End
            End
        End
    End

```

Note: Appendix A (Figure A.4) shows the flow chart for the flooding scheme.

Chapter 6 Implementation Issues and Experimental Results

6.1 Introduction

This chapter discusses the implementation of the prototypes based on the algorithms described in the previous chapter. The implementation focuses on performance issues already identified by other mobile agent implementations such as migration policies, coordination models and programming languages. The strategy followed for testing the prototypes comprises a set of test cases aimed to evaluate the impact of certain variables on the overall performance. In particular, the prototypes for the mobile agent search and the stationary agent (messages) search are tested under variable network size, network topology and number of agents.

6.2 Performance Issues

Several articles describe prototypes implemented in order to compare the performance between mobile agent applications and other approaches based on RPC-like mechanisms. In particular, in [BGM99] is presented an application for the retrieval of documents from a distributed collection of technical reports. In their experiments, reports are distributed across multiple machines, each of which is running D'Agents system. Client agents consult yellow pages to determine the location of the document collections. Once the

agent has the list of document collections, it must interact with the stationary agents, which serve as an interface between D'Agents and the collection. If the query requires only a few operations per document collection, the agent simple makes RPC-like calls across the network. If the query requires several operations per document collection, or if the operations involve large amount of intermediate data, the agent sends out child agents that travel to the document collections and perform the query operations locally.

The experiments in [BGM99] were carried out in two computers with 200 MHz Pentium II processors running Slackware Linux. The performance tests illustrated that inter-agent communication involves significantly more overhead than RPC and that migration involves even more overhead than inter-agent communication. In their particular case, part of that overhead may have been produced because the agents were written in Tcl, which is a relatively slow scripting language. They also observed that the performance of the mobile agent approach improved when the number of queries increased.

In summary, it is concluded in [BGM99] that the technical report searcher agent generally will take longer to complete its query than the corresponding client/server solution primarily because 1) inter-agent communication across machines is slower than RPC, 2) inter-agent communication on the same machine is also slow, making migration less useful and 3) the application was written in Tcl. They also concluded, with less evidence that as network bandwidth, reliability and latency worsen the agent searcher will have better and better performance relative to the client/server solution.

To implement our prototype, all mobile and stationary agents are based on the TACOMA system. A more detailed analysis of the TACOMA performance can be found in [JS97]. This paper examines the delay associated with the execution of a *meet* operation. This is the main function in the TACOMA system and comprises all the necessary operations to move an agent from host to host. The experiments were conducted on two workstations HP C-160 running HP-UX 10.20 and executing TACOMA with the logging and security features disabled. The main operations measured were 1) Marshalling and sending a briefcase, 2) Receipt of briefcase by the firewall, 3) Creation of an execution environment for the meeting and 4) Sending a message back to the initiating agent.

After measuring the aforementioned parameters it was obtained a median of 21.675 microseconds for the meet operation compared to 5.05 microsecond using a Sun RPC for the same data. However, for an RPC, the server's execution environment is static and set up in advance while in Tacoma, this is one of the most time consuming operations. To improve the performance of the TACOMA system, it is recommended in [JS97] to copy, on demand, parts of the briefcase to the remote host as a high level technique. This can be exploited to speed up the activation of a target agent by sending data to that agent only when it is needed.

Based on the results presented in [BGM99] and the performance analysis of the Tacoma system presented in [JS97] we did not anticipate a very different result in our experiments. Even though the message passing approach is expected to perform better than the mobile agent approach, we would like to explore how far they are in our

particular case and how changes in topology may impact the final results. In addition, we intend to compare the performance within the same group of solutions, which means to compare the different agent based approaches and the message passing approaches and select the most suitable for each group as well as the most suitable topology.

6.3 Detailed Implementation

The prototypes implemented comprise mobile and stationary agents to perform the traversal algorithms presented in Chapter 5. The coordination models implemented combine direct coordination and whiteboard coordination techniques. Each node of the network contains the mobile agent platform and stationary agents such as: whiteboards, routers and blackboards. The routers are implemented as stationary agents. Mobile searcher agents can meet either locally or remotely with these routers and obtain information about any node of the network.

The whiteboards are implemented as TACOMA cabinets. These cabinets are persistent structures where travelling agents can register upon arriving at a site. TACOMA provides an API to create and manage this kind of structures. The blackboards, on the other hand, are implemented as independent TCP/IP multithreaded servers that agents can contact to inform about the creation of new agents and to inform about the completion of a partial search.

According to the performance test results presented in previous sections, the programming language selected seems to be one of the factors affecting the overall performance. For this reason, the agents implemented in our prototype are written in C/C++. Moreover, the agent migration mechanisms are relatively simple in TACOMA, leaving all the agent's internal state management to the programmer. This approach as well as the availability of the system source code provides an extra flexibility for experimentations.

All the experiments presented herein, were developed in the Graduate Lab of the School of Computer Science. The agent platform used was TACOMA 1.2 compiled and run on the Sigma Network (Linux RedHat 6.1 machines) and the Ultra Network (SunOS 5.7 Generic_106541-06 sun4u sparc SUNW, Ultra-5_10). The blackboard associated to each node was implemented as a multi threaded application using the Linux version of POSIX threads. Routers and whiteboards were implemented as stationary agents and placed into TACOMA's system agent directory. The size of the information carried by the agent (DATA folder) was the same for the tree algorithms. Agents performed a dummy search at each site consisting basically of opening a file a reading the content. Since all the tests were carried out in an actual networked environment, the limitations on the number of machines involved in the experiments were determined by the number of nodes available at the Graduate Lab of the School of Computer Science.

6.4 Experimental Results

The prototypes are tested to measure the performance of the different approaches implemented. The objective is to obtain sufficient information to compare the performance between the mobile and stationary agents as well as the performance within the same category (agents or messages). In the experiments, agents are injected from any node of a network with the purpose of performing a distributed search. The launching application or initiator will measure the time in milliseconds (calling the function *fime*) before injecting the first agent and after the last agent has arrived. The difference between these two measurements will be reported as the execution time for the entire search.

The experiments attempt to measure the impact of certain variables on the performance of the algorithms. For each variable (size of the network, number of agents and network topology) is designed a test case. The following sections explain the different test cases and the results obtained. All the experiments were carried out in the absence of simulated failures and under normal circumstances. The computers were used simultaneously with other users and applications and no special care was taken to guarantee an exclusive access to computer resources or network during these experiments. A complete listing of the execution times for each experiment can be found in the Appendix B.

6.4.1 Test Case 1: Variable Network Size

This test case is designed to evaluate the performance of the mobile agents solutions and message passing solutions on a network with variable size. For each solution evaluated, a prototype that implements the algorithms presented in Chapter 5 is executed in a network with a tree topology. The number of neighbours for each node is constant and equals to 2 ($K=2$) and the size of the network (N) will be increased starting from $N=3$ until $N=30$.

The objective of this test is to observe whether or not there is any predicted behaviour in terms of performance as the number of nodes in the spanning tree increases. Moreover, it would be desirable to identify which algorithm performs better within its category.

The Figure 6.4.1 and Figure 6.4.2 (Appendix B, Table 1.....10) show the average execution times for the single and multiple travelling agent as well as the single travelling message and the flooding of messages.

Figure 6.4.1. Average execution times for mobile agents.

Figure 6.4.2. Average execution times for stationary agents.

As anticipated, the message passing solutions perform much better than the mobile agent solutions. It is important to note that both approaches are implemented on Tacoma 1.2 as communication platform in one case for agents and in the other for messages. All components, blackboards, whiteboards and routers are implemented for both approaches. The main difference resides in the agent itself. For the message passing approach the agents are stationary and they only exchange a briefcase containing the partial search results (information gathered at each node) and other information necessary for the coordination. Consequently, the agent's code is not transmitted nor it is necessary to activate the agent at each host.

Having confirmed this predictable result, it is interesting to examine how well the mobile agents solutions perform as the size of the network changes and to determine the most suitable algorithm to adopt in case of applying a mobile agent approach. As shown in the Figure 6.2.1, the implementation for the single travelling agent performs better for smaller network sizes. As of $N=18$ it is observed a better performance of the flooding algorithm. This is probably because for this case the advantages of having parallelism (multiple agents travelling simultaneously) surpass the overhead of coordinating the agents.

Even though the maximum number of nodes tested is not considerable large, it is observed a constant behaviour and perhaps increasingly better of the flooding or multiple agent approach over the single travelling agent. Similar results were obtained for the message passing approach. Although not as clear in the message passing approach as in

the mobile agents, a multiple cooperating agents scheme seems to be the most efficient mechanism to implement a distributed search on the spanning tree.

6.4.2 Test Case 2: Variable Number of Neighbours

This test case intends to examine the performance of the proposed solutions when the number of neighbours increases. The prototypes are executed on the logical spanning tree topology with 20 nodes ($N=20$, N : network size), starting with a binary tree and incrementing by 2 the number of neighbours ($K=2,4,6,\dots$) while maintaining the size of the network constant. Consequently, the extreme cases for this experiment will be a binary tree and a star topology respectively.

The Figure 6.4.3 and Figure 6.4.4 (Appendix B, Tables 11,.....19) show the average execution times for the single and multiple travelling agent as well as the single message and the flooding of messages.

Figure 6.4.3 Mobile agents with variable number of neighbours.

Figure 6.4.4 Stationary agents with variable number of neighbours.

Contrary to our expectations, the performance of any of the algorithms does not improve when the number of neighbours is increased. For the mobile agent approach, the only case when the flooding performs better than a single travelling agent is for the case of a binary tree. This result is in accordance with the first test case, where for N greater or equals to 18 nodes and two neighbours per node, the flooding offers better execution times. In the flooding scheme, as the number of neighbours per node increases, so does the number of agents and the need for coordination and interactions. This seems to produce an overhead that although does not increase dramatically, it prevents this solution from being better than a single travelling agent.

On the other hand, the implementation of the flooding algorithm for the message passing approach always performs better than the single travelling message. Even though the number of simultaneous messages also increases when the number of neighbours does, there is no need for code migration or to establish the agent execution environments at each node. Thus, parallelism offers an advantage over the single message but the increasing need for coordination and interaction seems to influence the overall performance enough to prevent the entire system from performing better when the number of neighbours increases.

6.4.3 Test Case 3: Different Topologies

This test case intends to execute the single travelling agent implementation and the flooding implementation in several networks with different topologies. As in previous test cases, the tests are grouped by agent-based approaches and message passing approaches, respectively. The network size is constant and equals 16 ($N=16$). The topologies compared are ring (bi-directional and unidirectional), hypercube and a binary tree.

A Ring is a network topology or circuit arrangement in which each device is attached along the same path to two other devices to two other devices, forming a path in a shape of a ring. Each entity in the ring has a unique address and the information flow may be unidirectional or bi-directional. The ring topology is very important for distributed

computing because of its simple and symmetric structure. Many algorithms are formulated for rings and then generalized to other classes of networks.

A hypercube, on the other hand, can be seen as an extension of a single rectangle. A hypercube of order N means that it has N dimensions. Consequently, the number of nodes would be 2^N and the number of connections per node are N . In general, a hypercube of order N may be constructed from two hyper-cubes of order $N-1$ by linking all pairs of corresponding nodes. For our experiment a logical hypercube of order 4 (16 nodes) was constructed.

The Figures 6.4.5, 6.4.6, 6.4.7 and Figure 6.4.8 show the average execution times of the two solutions (single travelling agent and the flooding) applied to the three topologies.

Figure 6.4.1 Average execution times for a single travelling agent

Figure 6.4.2 Average execution times for multiple travelling agents.

Figure 6.4.3 Average execution times for a single travelling message.

Stationary Agents (Message Passing)

Figure 6.4.4 Average execution times for multiple messages.

For the case of a single travelling agent, the execution times are similar for the tree and the ring topologies. This is the case for a single travelling agent as well as for a single travelling message but not for the hypercube, which shows the worst execution times for both algorithms in the two approaches.

The multiple travelling agents or flooding also presents the hypercube as the worst topology. In this case, the bi-directional ring performs better as expected, thanks to the parallelism because there are two agents travelling in opposite directions. For the hypercube, even though there is parallelism as well, it seems that the overhead required to clone agents and verification of multiple visits as well as communication with the

blackboard, constitutes an extra overhead eliminating any additional advantage that parallelism may offer.

For this experiment, the tree topology selected only has two neighbours per node (binary tree). This topology, according to the second test case, is the most efficient among the trees tested. In addition, for this particular experiment the tree has 18 nodes, ($N=18$) which means two nodes more than the other topologies. Even under this circumstances, the tree performed as efficiently as the bi-directional ring and on some occasions, even better. The appendix B shows the tables (Table 20, 21 and 22) with the execution times for the algorithms and their averages for this particular test case

6.5 Summary of Experimental Results

Unlike other experimental results ([RD99], [BGM99]) where under certain circumstances the mobile agent solutions outperform the message passing implementations, in our experiments, the stationary agents performed better than the mobile agent for all the test cases. In the test of variable network size, when the network sizes increases, the multiple agent or messages offers a better performance than the single travelling agent or message. Multiple agents introduce a desirable level of parallelism in the search that for larger number of nodes seems to counteract any negative impact produced by the overhead of coordinating the agents.

According to this result seems logic to believe that the larger the number of agent the better is the performance of the search. Unfortunately, the results of the second test case indicate the contrary. As the number of agents increases, so does the need for coordination and interactions. This situation seems to produce an overhead that although does not affect dramatically the performance of the system, it is sufficient to prevent it from performing better than the single travelling agent.

The results of the last test case show that the topology of the network does impact the overall performance. In this particular case, the tree topology proposed showed the best performance among the topologies tested. In particular, the binary tree offered the best response times when compared with other trees with larger number of neighbours per node.

Chapter 7 Conclusions

7.1 Thesis Summary

Throughout this work we have examined the suitability of an agent-based approach to solve the distributed search or information retrieval problem. Several research papers related with this topic and their solutions have been surveyed. Some characteristics of agent systems such as: migration mechanism, languages supported and fault tolerance, have also been discussed as well as some examples of current mobile agent platforms. By knowing how to characterize and classify the agent systems available, a designer will be able to make a more informed choice about the most suitable platform according the application requirements.

There are some issues, which are important to consider when implementing a mobile agent solution. Agents need to coordinate and synchronize their work and also need to communicate. Coordination can be direct, meeting oriented, or based on whiteboards or tuples. There is no evidence of a best coordination mechanism in the literature. The selection depends on the application and sometimes a combination of several techniques, offers the most satisfactory results.

We have presented a model to implement an agent based distributed search as an alternative to the well-known message passing approaches. The model comprises several

components, which are responsible for the exchange and coordination of cooperating agents. Along with the mobile agent platform, our architecture defines mobile and stationary agents. The coordination of agents is achieved by using blackboards, whiteboards and routers. Figure 7.1.1 summarizes the architecture of the model proposed along with all the components involved.

Figure 7.1.1 System Architecture.

Several experiments were carried out using prototypes that implement the algorithms for a single travelling agent as well as for multiple agents. The corresponding message passing algorithms were also implemented to facilitate the comparison between the two approaches. The different test cases showed that the flooding algorithm performs better than the single agent when the size of the network increases. Multiple agent solutions exploit the fact that the search space can be divided among several cooperating agents introducing a desirable level of parallelism. Furthermore, the tree topology proposed,

performed better than other topologies tested for a constant network size. In particular, the binary tree offered the best results when compared to other tree structures with larger number of neighbours per node.

The experiments were carried out assuming an absence of failure (communication failures, node crashes, etc.). Degradation in performance is predicted in the case of multiple node failure due to an intensive use of the global routers. In the case of a failure, the agent contacts the local agent router or the nearest router previously visited by it. This mechanism provides a fair amount of confidence because the agent initially tries a local meeting, which is more likely to be successful. If the local router does not contain a global routing table, then it is probable that the previous router visited is still active. If so, the request is directed to it. To improve the mechanism, a chain of routers can be established, and the agent can explore the chain until finding an active router. This solution may increase the size of the agent, which is not desirable. Another alternative may be to store such a chain into the local router information. Consequently, each local router can provide a list of secondary global routers.

7.2 Future Work

Further improvements can be made to the mechanism used to communicate the launching application with the arriving agents. Currently, the calling application blocks after sending the first agent to the network. When the agent returns to the initiator, it raises a signal that is captured by the calling application. Another mechanism, also implemented,

consists of agents writing their partial search results into a pipe upon arriving to the initiator. The initiator will read the results from the pipe until the last agent has arrived. This condition is verified by querying the blackboard.

The migration process utilized in the prototype is a-synchronous. A synchronous mechanism, also supported in TACOMA, is very costly because it ties up resources in each computer that the agent visits. Furthermore, implementing a flooding algorithm based on synchronous migration would be excessively demanding in terms of computational resources.

The creation of a component responsible for sensing the network connections and dynamically reassigning the local and global routing tables is a desirable enhancement. Besides, future modifications might explore algorithms to propagate more efficiently changes on the global routing tables and creating new tables dynamically.

So far, only experiments involving the agent environment have been carried out. The integration of the search facility with a distributed computing environment as proposed by the Globus¹ project might be part of future work. The integration with the Globus's distributed file system component (GASS)[BF99] would be beneficial to achieve location transparency of remote resources. Consequently, applications built using the new modified version of GASS will be able to locate resources in a simpler manner.

¹ <http://www.globus.org>

Current mobile agent platforms do not offer a better performance compared to message passing approaches, perhaps, because the cost of moving the entire agent and applying coordination techniques is still too high. These mobile agent platforms are still far from their optimal performance. Agents have become more attractive these days and with the evolution of agent systems, the simplicity of agent based solutions and their flexibility, mobile agent computing will be more and more used to solve problems requiring distributed computing.

References

- [BF99] J. Bester, I. Foster, C. Kesselman, J. Tedesco, S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems," in *Proceedings Sixth Workshop on I/O in Parallel and Distributed Systems*, May 1999.
- [BG99] B. Brewington, R. Gray, K. Moizumi, "Mobile Agents in Distributed Information Retrieval," in *Intelligent Information Agents*, 1999, pp. 355-395.
- [CF99] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," in *Journal of Network and Computer Applications*, 1999.
- [CL00] G. Cabri, L. Leonardi, F. Zambonelli, "Mobile-Agent Coordination Models for Internet Applications," in *Computer*, Feb. 2000, pp. 82-89.
- [CT98] P. Ciancarini, R. Tolksdorf, F. Vitali, D. Rossi, A. Knoche, "Coordinating Multi-Agent Applications on the WWW: A Reference Architecture," in *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, May 1998.
- [DW99] D. Deugo and M. Weiss, "Agent Patterns, Tutorial," *International Symposium on Agent Systems and Applications and The International Symposium on Mobile*

Agents and the Dartmouth Workshop on Transportable Agents (ASA/MAT99), Oct. 1999.

- [FK97] B. Falchuk and A. Karmouch, "A Mobile Agent Prototype for Autonomous Multimedia Information Access, Interaction and Retrieval," in *Proceedings Multimedia Modelling '97*, Nov. 1997.

- [JJ97] K. Jacobsen, and D. Johansen, "Mobile Software on Mobile Hardware. Experiences with TACOMA on PDAs," Technical Report 97-32, Department of Computer Science, University of Tromsø, Norway, Dec. 1997.

- [JM98] D. Johansen, K. Marzullo, K.J. Lauvset, "An Approach towards an Agent Computing Environment," in *ICDCS'99 Workshop on Middleware*. Department of Computer Science, University of Tromsø, Norway, Dec. 1998.

- [JR94] D. Johansen, R.V. Renesse, F. B. Schneider, "Operating system support for mobile agents," in *Proceedings of the 5th. IEEE Workshop on Hot Topics in Operating Systems*, USA, May 1995.

- [JR95] D. Johansen, R.V. Renesse, F. B. Schneider, "An Introduction to the TACOMA Distributed System Version 1.0," Technical Report 95-23, Department of Computer Science, University of Tromsø, Norway, Jun. 1995.

- [JS97] D. Johansen, N. P. Sudmann, R.V. Renesse, "Performance Issues in TACOMA," in *3rd. Workshop on Mobile Object Systems, 11th European Conference on Object-Oriented Programming*. Finland, Jun. 1997.

- [L98] L. T. Chen. "AgentOS: The Agent-based Distributed Operating System for Mobile Networks," in *Crossroad, ACM Student Magazine*, 1998.

- [MC98] K. Moizumi and G. Cybenko, "The Travelling Agent Problem," in *Mathematics of Control, Signals and Systems*. Jan. 1998.

- [OK99] A. Outtagarts and M. Kadoch, "Client-Server and Mobile Agent: Performances Comparative Study in the Management of MIBs" in *Proceedings of the First International Workshop on Mobile Agents for Telecommunication Applications (MATA '99)*, 1999, pp. 69-81.

- [P98] H. Peine, "Security Concepts and Implementation in the Ara Mobile Agent System," in *Proceedings of WETICE '98*, Jun. 1998.

- [PS97] H. Peine and T. Stolpmann, "The Architecture of the Ara Platform for Mobile Agents," in *Proc. of the First International Workshop on Mobile Agents MA'97, Lecture Notes in Computer Science*, No. 1219, 1997.

- [RD99] G. Rubinstein and M.G. Duarte, "Evaluating the Performance of Mobile Agents in Network Management." in *Global Telecommunication Conference (BLOBECOM'99)*, Vol. 1a, 1999, pp. 386-390.
- [RG96] D. Rus, R. Gray, D. Kotz. "Autonomous and Adaptative Agents that Gather Information," in *Proceedings AAAI'96 International Workshop on Intelligent Adaptative Agents*, 1996.
- [W95] W. Schoenfeldinger, "WWW Meets Linda: Linda for Global WWW-Based Transaction," in *World Wide Web J.*, Vol. 1, No. 1, Dec. 1995, pp. 259-276.
- [WJ95] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *The Knowledge Engineering Review*, Vol. 10, No. 2, 1995, pp. 115-152.

Appendix A Flow Charts

Figure A.1 The Single Travelling Agent

Figure A.2 The Single Travelling Message

Figure A.3 Multiple Travelling Agents

Figure A.4 The Multiple Travelling Messages

Appendix B Test Results

Table 1: Variable Network Size (N=3)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
1256	1276	2199
1272	1262	2434
1269	1270	2369
1256	1252	2494
1257	1276	2209
1277	1245	2474
1270	1261	2038
1259	1247	2250
1274	1275	2045
1254	1278	2379
1273	1290	2250
1254	1244	2436
1296	1252	2196
1276	1249	2154
1283	1284	2214
1252	1251	2271
1266	1244	2242
1283	1256	2236
1274	1258	2154
1249	1250	1675
1272	1271	2241
1264	1260	2263
1271	1270	2273
1290	1243	2156
1251	1252	1673
1248	1257	1695
1262	1282	2141
1272	1256	2195
1250	1259	1684
1252	1274	1675
1266.07	1261.47	2157.17

Stationary Agents		
Depth First	Breadth First	Flooding
59	64	84
61	62	81
64	61	77
58	58	81
61	62	73
63	62	77
62	58	76
57	58	78
63	59	78
63	59	79
63	64	81
62	60	76
63	57	81
59	58	78
60	61	77
60	54	74
58	62	81
55	58	76
64	64	80
61	62	80
63	59	78
56	58	79
62	64	74
62	54	82
58	60	75
61	54	76
60	54	77
59	60	78
58	57	78
61	56	80
60.53	59.30	78.17

Mobile Agents

Stationary Agents

Table 2: Variable Network Size (N=6)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
2484	2514	3368
2488	2536	3338
2578	2497	2796
2540	2588	3358
2525	2533	3581
2561	2493	3276
2504	2566	3266
2518	2501	3513
2507	2505	3436
2568	2492	3530
2511	2590	2755
2517	2512	2798
2509	2524	3330
2566	2496	3325
2507	2518	3272
2521	2570	3346
2560	2511	3343
2513	2508	2757
2515	2507	2762
2542	2580	3559
2554	2494	3373
2543	2530	3337
2572	2510	2803
2508	2575	3264
2528	2509	3311
2516	2522	3330
2582	2582	3297
2512	2522	2787
2503	2529	3291
2571	2527	2774
2530.77	2528.03	3209.20

Stationary Agents		
Depth First	Breadth First	Flooding
102	122	121
112	122	117
102	119	118
104	124	117
100	124	118
131	121	112
102	121	115
103	124	116
108	118	125
109	122	116
109	123	119
102	125	115
101	115	116
106	120	118
104	116	119
108	121	116
106	120	117
106	118	111
102	117	116
110	114	116
106	122	118
102	120	116
99	119	119
112	119	119
127	116	119
125	121	109
128	121	115
127	119	117
126	113	117
127	116	117
110.20	119.73	116.80

Mobile Agents **Stationary Agents**

Table 3: Variable Network Size (N=9)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
3731	3872	4566
3733	3772	4820
3874	3782	4739
3775	3792	4737
3865	3874	5019
3784	3777	5077
3781	3788	5173
3880	3800	5008
3777	3875	4917
3855	3787	4847
3766	3923	5258
3875	3786	5217
3768	3869	5193
3779	3800	4981
3944	3772	5210
3769	3795	5035
3777	3911	4797
3794	3841	4891
3832	3830	4809
3779	3939	4782
3862	3788	5035
3772	3833	4914
3782	3880	4733
3866	3768	4897
3799	3858	4778
3787	3771	5060
3785	3912	4754
3856	3787	4910
3794	3804	4995
3766	3879	4960
3806.90	3828.83	4937.07

Stationary Agents		
Depth First	Breadth First	Flooding
226	176	150
222	182	160
183	190	147
177	195	175
180	172	146
217	171	176
184	171	167
177	176	187
177	184	165
178	189	151
184	181	152
174	189	190
223	187	194
165	188	147
166	183	167
171	183	166
167	181	147
169	182	153
167	180	164
220	187	152
174	187	149
169	188	158
173	181	161
165	180	153
162	180	157
162	188	159
211	181	154
155	187	155
179	184	160
165	186	149
181.40	182.97	160.37

Mobile Agents **Stationary Agents**

Table 4: Variable Network Size (N=12)

(Running time: milliseconds)

Mobile Agents			Stationary Agents		
Depth First	Breadth First	Flooding	Depth First	Breadth First	Flooding
5046	5203	5824	247	235	182
5032	5037	5814	261	234	182
5549	5038	6001	258	272	181
6567	5166	8407	291	242	184
6100	5076	5767	270	277	190
6450	5198	5873	237	249	185
6295	5041	5999	246	256	176
5909	5180	5746	283	232	182
5066	5024	5798	244	277	183
5080	5054	5825	267	243	198
5201	5194	7341	271	269	188
5065	5066	5826	236	289	402
5053	5076	5826	234	249	333
5271	5191	6184	251	245	239
5097	5051	5842	268	256	179
5156	5063	6059	239	281	181
5125	5161	5852	233	257	214
5053	5046	6330	241	254	199
5280	5014	5802	242	249	188
5055	5293	5823	242	260	195
5066	5104	5989	261	251	179
5179	5181	5806	236	274	208
5060	5166	6411	236	245	189
5047	5182	5755	239	257	195
5183	5262	5687	247	264	196
5048	5050	5824	237	256	188
5045	5740	5807	233	260	183
5165	5184	5871	258	259	194
5037	5594	5730	234	255	234
5068	5049	5748	240	251	192
5311.60	5156.13	6018.90	249.40	256.60	203.97

Mobile Agents **Stationary Agents**

Table 5: Variable Network Size (N=15)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
7944	6300	7261
8666	6578	7772
7818	6428	7344
7756	7342	6977
8094	7215	7367
6349	7479	8262
6456	7265	8783
6444	7368	8105
6364	7491	9961
6278	6328	6899
6513	6294	6675
6294	6453	7072
6298	6386	7439
6497	7382	7383
6479	7262	7399
6329	7265	6697
6294	6251	6822
6438	6476	6825
6350	6338	7921
6293	6279	6819
6460	6566	6659
6278	6283	7102
6312	6293	7635
6299	6455	7574
6463	7431	7512
6480	7625	7748
6293	7298	7442
6429	7338	6835
6630	7333	7034
6490	7541	6986
6669.60	6878.10	7410.33

Stationary Agents		
Depth First	Breadth First	Flooding
337	313	451
388	430	228
325	883	215
325	337	661
305	332	237
327	318	238
312	319	389
319	354	240
330	332	229
305	330	221
306	432	336
680	578	237
449	554	229
487	694	239
599	639	240
604	607	236
330	713	411
311	339	687
300	310	227
282	324	233
335	311	519
384	313	253
334	307	642
291	303	441
300	313	225
286	304	226
290	318	547
322	300	642
297	316	430
328	306	651
359.60	407.63	358.67

Mobile Agents **Stationary Agents**

Table 6: Variable Network Size (N=18)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
7634	7688	7852
7947	7637	7817
7616	7687	7582
7977	7689	7661
7645	7633	7723
7953	7899	7697
7675	7656	7710
7628	7691	7942
7724	7587	7664
7660	7671	7800
7882	7771	7771
7630	7586	7474
7639	7674	7904
7603	7641	7374
7601	7623	7828
7793	7576	7608
7598	7604	7682
7654	7658	7944
7632	7652	7981
7748	7656	7815
7601	7642	7461
7624	7644	7877
7744	7664	7826
7804	7673	7787
7665	7528	7573
7610	7557	7809
7741	7669	7827
7752	7617	7298
7704	7649	7566
7580	7717	7816
7702.13	7654.63	7722.30

Stationary Agents		
Depth First	Breadth First	Flooding
390	364	281
383	364	485
358	371	244
363	371	429
392	370	263
371	371	263
356	365	461
365	371	246
377	354	239
364	356	242
365	354	251
364	355	262
358	361	266
361	356	237
353	361	330
360	366	244
366	352	254
358	363	244
357	375	252
358	364	262
348	374	241
350	371	255
355	373	259
366	379	252
350	371	250
344	363	251
360	378	250
360	380	253
372	373	259
362	378	241
362.87	366.80	275.53

Mobile Agents **Stationary Agents**

Table 7: Variable Network Size (N=21)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
11552	11365	10200
12473	11135	13566
10520	11570	9650
10445	10517	9727
10629	10527	10024
11181	11179	10369
11098	10427	11002
10303	11154	9701
10551	11448	9822
10463	11247	9573
11271	10523	9071
11713	10400	13474
10558	10520	9351
10356	10825	9492
11541	10409	9080
11719	10540	9158
10455	10552	9585
11021	11889	10266
10378	11269	11100
10658	12101	9304
10373	11118	8906
10482	12173	10260
10371	10492	10303
11205	10938	10860
11083	11177	9185
10524	10387	9555
11196	11497	8836
11199	10445	10191
11793	10419	11508
10659	11460	10391
10925.67	10990.10	10117.00

Stationary Agents		
Depth First	Breadth First	Flooding
870	889	485
881	895	959
864	862	471
870	844	803
883	864	769
885	843	898
862	900	886
923	848	1082
904	865	926
859	830	748
866	837	866
857	838	931
895	946	916
913	846	972
895	857	889
876	924	887
885	828	921
884	875	960
876	839	753
860	801	968
865	923	891
867	866	944
858	900	494
853	859	699
918	859	479
980	813	933
904	905	800
933	849	692
852	846	923
851	880	885
882.97	864.37	827.67

Mobile Agents **Stationary Agents**

Table 8: Variable Network Size (N=24)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
13328	13198	10272
13017	12542	10334
13045	12496	10396
13349	12436	9961
13081	12310	10210
13786	13001	9988
13701	12438	13335
13182	12248	10644
12334	12245	10017
13749	12425	10632
12899	12218	11287
13020	13044	9988
12516	12246	10806
12387	12295	9799
12329	12492	10618
12707	12637	10268
12781	13491	10549
12801	12212	10604
13154	12343	9687
13474	12279	10934
13329	12448	9784
12370	12169	10506
13093	12958	10046
12828	13020	9899
13362	13058	9729
12748	12910	10652
12927	12937	10297
13468	12391	9662
13279	12308	9799
12433	12419	10201
13015.90	12573.80	10363.47

Stationary Agents		
Depth First	Breadth First	Flooding
1051	1092	512
1057	1014	884
1112	1024	869
1050	1014	842
1051	1026	691
1033	1023	809
1059	1063	600
1034	1055	1119
1111	1025	997
1066	1011	540
1046	1030	922
1042	1009	720
1057	1036	569
1110	1061	862
1044	1048	807
1086	1021	824
1099	1025	826
1072	1064	837
1086	1011	795
1085	1009	898
1089	1061	783
1091	1008	974
1063	1040	961
1093	1059	671
1082	1034	728
1064	1035	956
1093	1063	793
1064	1074	531
1089	1042	752
1085	1047	684
1072.13	1037.47	791.87

Mobile Agents **Stationary Agents**

Table 9: Variable Network Size (N=27)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
19094	20845	11846
19148	19910	11514
19920	20664	11357
19095	18809	11036
18403	19750	16630
20351	20672	10996
18686	20145	11336
19045	20671	11529
19880	20033	11024
19008	19708	11254
20039	22588	11337
18911	21203	10943
18974	21180	11082
18862	19393	11103
19383	19560	11672
19630	19143	11421
19655	19615	11112
19152	20738	12075
19144	20215	11575
19811	19308	12166
19361	19708	11120
20387	20081	11651
20140	19575	11289
19208	19711	11497
20495	20256	12159
19711	18995	12146
19619	19621	11073
19881	21190	11742
19436	19925	11297
19266	20244	11577
19456.50	20115.20	11618.63

Stationary Agents		
Depth First	Breadth First	Flooding
2071	2491	1223
2311	2177	786
2007	2364	1034
2415	2026	908
2036	2120	1102
2178	2187	1035
1873	2217	946
2311	2084	917
2421	2116	1108
2262	1981	1299
2312	1897	753
2455	2020	1404
2201	2339	895
1941	1600	960
2318	2075	737
2287	2245	1263
2036	2236	1005
2347	2049	840
2005	2311	997
2181	1971	984
2357	2211	1532
2220	2091	1058
2046	1785	821
2230	2067	1009
2341	1831	1089
2364	1699	910
2270	2344	996
2166	1978	1375
2235	2322	1061
2015	2126	957
2207.07	2098.67	1033.47

Mobile Agents **Stationary Agents**

Table 10: Variable Network Size (N=30)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
19736	23039	14802
18567	20771	13168
18107	22967	12951
17775	17652	13315
17796	20843	13044
16619	17916	14839
16612	20570	11757
17753	20688	15535
20830	16882	18480
17838	17390	12108
20976	17550	12146
17000	16635	15540
19855	17872	11764
17250	18412	11645
18221	16954	12059
18946	19318	12022
20881	21200	11808
16545	18592	12293
16205	18043	11729
15893	19635	11922
16528	17170	13761
20604	17902	17118
16381	16220	16648
16968	21424	11525
18758	18749	11626
18333	16968	12035
16261	18264	11757
17967	21060	11454
16547	16276	12732
16372	17980	15475
17937.47	18831.40	13235.27

Stationary Agents		
Depth First	Breadth First	Flooding
1565	1554	983
1539	1531	760
1508	1520	971
1517	1380	992
1495	1537	790
1873	1508	946
1550	1790	969
1290	1463	733
1593	1462	757
1508	1472	1007
1558	1573	1104
1428	1360	962
1893	1504	1035
1654	1518	971
1542	1459	779
1567	1484	1001
1602	1528	923
1597	1643	982
1517	1492	744
1801	1474	965
1644	1487	801
1551	1455	1029
1480	1525	1016
1585	1558	1178
1608	1487	992
1529	1575	947
1561	1442	1207
1335	1595	751
1579	1531	973
1456	1451	1169
1564.17	1511.93	947.90

Mobile Agents **Stationary Agents**

Table 11: Variable Number of Neighbours (k=2)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
8291	8598	8416
8258	8408	8352
8296	8382	8135
8661	8610	8572
8411	8427	8350
8397	8574	8362
8439	8437	8339
8605	8363	7725
8493	8349	8182
8408	8377	8118
8418	8357	8511
8558	8592	8177
8400	8366	8773
8445	8397	8664
8442	8479	8399
8587	8602	8078
8447	8388	8324
8380	8400	8183
8402	8376	8374
8400	8380	8412
8424	8607	8538
8560	8464	8312
8597	8375	8151
8437	8521	8577
8378	8390	8355
8461	8565	8130
8414	8375	8107
8407	8442	8164
8399	8372	8199
8429	8386	8414
8441.47	8445.30	8313

Stationary Agents		
Depth First	Breadth First	Flooding
419	419	255
414	419	252
417	412	259
422	414	246
416	416	257
418	417	251
406	416	252
419	418	310
416	409	261
419	416	262
417	413	314
416	410	254
415	416	270
418	417	253
403	415	253
413	416	295
410	413	262
410	412	254
410	422	243
404	423	264
412	424	255
404	422	380
412	423	262
408	420	252
408	418	262
413	418	262
406	419	258
407	418	257
410	418	260
412	420	260
412.47	417.10	265.83

Mobile Agents **Stationary Agents**

Table 12: Variable Number of Neighbours (K=4)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
	8669	9894
	8382	9909
	8395	9876
	8604	10064
	8706	10624
	8640	10096
	8392	10086
	8393	10441
	8709	10463
	8382	10094
	8406	9823
	8595	10290
	8460	9992
	8375	9874
	8417	10461
	8603	10090
	8372	10345
	8385	9884
	8622	10096
	8419	10034
	8672	10032
	8406	10571
	8492	10392
	8415	10263
	8761	10544
	8547	10095
	8574	10482
	8390	10084
	8603	10057
	8620	10062
	8513.53	10167

Stationary Agents		
Depth First	Breadth First	Flooding
	435	331
	436	294
	436	329
	437	310
	437	338
	438	354
	432	334
	423	387
	425	338
	425	339
	422	349
	429	320
	429	330
	419	348
	414	333
	426	388
	416	331
	415	335
	418	323
	414	325
	426	328
	425	311
	410	337
	425	353
	426	322
	421	341
	423	327
	428	329
	424	333
	425	333
	425	335.00

Mobile Agents **Stationary Agents**

Table 13: Variable Number of Neighbours (K=6)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
	8278	10233
	8615	10273
	8396	10418
	4212	10617
	8629	10886
	8390	10451
	8412	10477
	8674	10667
	8397	10731
	8618	10649
	8404	10439
	8371	10753
	8585	10422
	8373	10847
	8458	10735
	8640	10439
	8476	10426
	8679	10378
	8437	10325
	8623	10461
	8451	10241
	8366	10735
	8700	10616
	8360	10625
	8397	10477
	8636	10472
	8429	10650
	8394	10649
	8573	10730
	8375	10461
	8344.93	10543

Stationary Agents		
Depth First	Breadth First	Flooding
	425	363
	418	361
	423	378
	421	344
	422	364
	427	384
	426	344
	426	365
	420	364
	422	377
	421	370
	421	377
	424	377
	420	373
	426	375
	422	365
	422	372
	423	380
	421	338
	424	355
	422	362
	418	377
	428	371
	422	391
	425	361
	424	365
	418	375
	412	380
	421	367
	407	384
	422	368.63

Mobile Agents **Stationary Agents**

Table 14: Variable Number of Neighbours (K=8)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
	8600	10922
	8509	12915
	8515	11239
	8548	11495
	8649	11024
	8525	11311
	8667	11429
	8699	10767
	8510	11128
	8482	11028
	8509	11431
	8535	10832
	8936	11055
	8857	11087
	8697	11093
	8625	11290
	8900	11129
	8718	11295
	8694	10927
	8741	10957
	8464	11013
	8643	11454
	8678	10844
	8604	12018
	8771	11123
	8636	11493
	8647	10753
	8617	11541
	8683	11282
	8779	10885
	8647.93	11225

Stationary Agents		
Depth First	Breadth First	Flooding
	541	430
	528	425
	446	553
	463	467
	462	439
	449	429
	451	413
	457	441
	449	456
	451	420
	450	425
	445	440
	446	434
	451	428
	448	403
	452	426
	419	411
	428	418
	454	419
	438	425
	447	458
	452	429
	453	458
	449	405
	463	407
	423	445
	428	394
	437	400
	446	410
	426	422
	452	431.00

Mobile Agents **Stationary Agents**

Table 15: Variable Number of Neighbours (K=10)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
	8499	11484
	8656	11374
	8744	11170
	8766	11231
	8526	11163
	8764	11243
	8683	10918
	8543	11528
	8639	11487
	8536	11439
	8526	11462
	8628	11290
	8706	11489
	8794	11101
	8767	11014
	8575	11195
	8784	11575
	8660	11161
	8663	11567
	8660	11534
	8690	11248
	8658	11567
	8574	11433
	8783	11426
	8626	11213
	8613	11466
	8577	11304
	8601	11225
	8656	11730
	8712	11468
	8654	11350

Stationary Agents		
Depth First	Breadth First	Flooding
	509	414
	510	415
	519	416
	519	421
	550	419
	533	453
	522	422
	527	470
	527	425
	513	440
	512	478
	526	419
	530	424
	526	490
	530	412
	526	424
	523	431
	528	432
	529	430
	525	438
	529	439
	513	429
	525	420
	517	416
	513	442
	520	430
	527	435
	521	426
	495	423
	525	431
	522	432

Mobile Agents **Stationary Agents**

Table 16: Variable Number of Neighbours (K=12)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
	8963	12113
	9591	11784
	8934	12055
	9489	11968
	9451	11964
	9657	12668
	9775	12017
	8817	11906
	8803	11780
	9061	11331
	9108	11769
	8770	11644
	8876	11588
	8770	11900
	8997	11332
	8866	11031
	8996	11531
	9161	11826
	8795	11207
	8848	11200
	8887	11143
	8739	11882
	8806	11674
	8734	11532
	8908	11865
	8920	11516
	8815	11362
	8745	11430
	8962	11669
	8834	11561
	9003	11675

Stationary Agents		
Depth First	Breadth First	Flooding
	530	449
	536	449
	537	449
	540	463
	523	449
	534	441
	526	446
	529	482
	534	438
	541	455
	539	418
	531	453
	522	441
	519	468
	526	433
	531	478
	535	441
	536	482
	531	481
	539	443
	531	463
	534	494
	529	505
	532	455
	536	441
	535	457
	536	452
	532	446
	525	427
	528	423
	532	454

Mobile Agents **Stationary Agents**

Table 17: Variable Number of Neighbours (K=14)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
	9550	11701
	8863	12301
	9653	11228
	8786	11392
	9551	11916
	9708	10978
	8906	11804
	9023	11683
	8773	11354
	8774	11399
	8747	12090
	9131	11399
	9000	11648
	8796	12153
	9283	11778
	9197	11644
	8834	11487
	9457	12154
	9171	11365
	9447	11987
	8808	11795
	8788	11281
	9077	11449
	8757	11372
	8995	12131
	8775	12147
	8794	11561
	8989	11106
	8892	12317
	9065	11722
	9053	11678

Stationary Agents		
Depth First	Breadth First	Flooding
	542	441
	539	439
	536	455
	537	448
	539	508
	537	459
	541	443
	535	442
	533	450
	537	442
	534	442
	534	445
	539	512
	526	461
	526	458
	527	462
	525	457
	525	454
	524	456
	522	449
	531	444
	534	474
	536	452
	527	453
	521	458
	527	451
	525	454
	525	488
	522	457
	534	487
	531	458

Mobile Agents **Stationary Agents**

Table 18: Variable Number of Neighbours (K=16)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
	12737	12277
	11476	11217
	11388	11201
	12442	11638
	11327	11757
	11938	11473
	11580	11259
	11457	11279
	12433	11903
	12015	11435
	11930	11827
	11628	11726
	11420	11559
	11569	11418
	11481	11500
	12046	11628
	11465	11500
	11970	11479
	11379	11621
	12168	11660
	11538	11565
	11615	11406
	11829	11635
	11581	11896
	11322	11058
	11282	11144
	11825	11792
	12430	11689
	11676	11605
	11600	11973
	11752	11571

Stationary Agents		
Depth First	Breadth First	Flooding
	978	708
	1028	470
	969	436
	1009	493
	1039	1012
	987	445
	1048	807
	1029	461
	993	469
	992	1075
	1053	495
	1024	461
	993	837
	1032	867
	1002	1089
	976	1059
	919	806
	1003	853
	1075	1005
	1060	833
	1056	856
	1089	853
	1032	912
	1076	440
	1021	856
	911	452
	1085	451
	1053	920
	1055	1027
	1052	436
	1021	729

Mobile Agents

Stationary Agents

Table 19: Variable Number of Neighbours (K=19)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
	11878	11280
	12482	11927
	10877	12028
	12030	11918
	11541	12247
	11380	11626
	11908	11779
	12150	12001
	12084	12018
	11497	12069
	11853	12089
	12543	11971
	11609	12227
	11937	12100
	11420	12080
	11573	12048
	11467	12138
	11549	12034
	11833	12014
	11707	11689
	11759	12562
	12302	12729
	11314	12324
	11607	12250
	11275	12024
	11560	11753
	11423	12387
	11672	11717
	12060	12281
	11801	11568
	11736	12029

Stationary Agents		
Depth First	Breadth First	Flooding
	990	520
	1096	524
	1052	548
	977	485
	1097	880
	967	843
	938	833
	1079	833
	1052	1099
	1089	624
	1060	1045
	1060	867
	1089	886
	982	843
	1006	1087
	1068	830
	1003	923
	1024	490
	979	475
	1052	911
	1051	491
	949	874
	1096	531
	1055	885
	1071	490
	982	1001
	1027	484
	1046	915
	1087	876
	986	450
	1034	751

Mobile Agents **Stationary Agents**

Table 20: Bi-directional Ring (N=16)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
	8088	6846
	7277	7595
	8051	6845
	7165	7613
	7206	7357
	8037	7879
	7346	7965
	7250	6806
	7418	6995
	7180	8037
	7379	7569
	7292	7392
	7367	7578
	7258	7766
	7362	6855
	7333	7084
	7181	6931
	7176	7511
	7466	7803
	7898	7525
	7366	7069
	7174	7961
	7180	7817
	7163	7620
	7985	7081
	7962	7699
	8586	7641
	8555	7699
	7214	6717
	8096	7362
	7534	7421

Stationary Agents		
Depth First	Breadth First	Flooding
	388	242
	374	257
	419	279
	365	267
	365	249
	371	250
	408	248
	368	243
	369	264
	392	270
	365	245
	370	257
	373	244
	369	239
	370	247
	369	256
	381	275
	384	256
	386	313
	377	242
	383	272
	377	262
	383	249
	379	239
	378	260
	388	238
	394	253
	398	251
	363	221
	394	225
	380	254

Mobile Agents **Stationary Agents**

Table 21: Unidirectional Ring (N=16)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
	8880	9459
	8096	10506
	8752	11089
	7439	9684
	8239	9626
	7379	10379
	8080	9573
	7647	10248
	8123	9935
	7221	9684
	7414	9717
	7884	9554
	7334	9884
	8310	9680
	7361	10528
	8026	9919
	7303	9613
	7989	9657
	7247	9721
	7207	9747
	7280	10395
	7362	10332
	7266	10861
	7339	9779
	7284	9662
	7301	9570
	8011	9544
	7307	9697
	7731	9858
	7350	10317
	7672	9941

Stationary Agents		
Depth First	Breadth First	Flooding
	336	896
	368	742
	365	862
	332	950
	328	960
	341	931
	329	893
	336	958
	332	859
	400	911
	355	991
	358	916
	351	722
	398	767
	348	946
	363	924
	361	753
	404	940
	358	946
	383	743
	362	974
	385	756
	378	735
	367	743
	388	713
	375	997
	368	749
	365	986
	378	975
	363	957
	363	873

Mobile Agents **Stationary Agents**

Table 22: Hypercube (N=16)

(Running time: milliseconds)

Mobile Agents		
Depth First	Breadth First	Flooding
	8693	11018
	8731	14509
	9115	14212
	11593	11291
	8988	16511
	12393	13267
	11839	12599
	8586	10871
	10382	12447
	8338	11294
	9097	10771
	9849	10618
	9511	10555
	8602	10705
	10306	10425
	11785	10121
	9021	10839
	8950	11726
	9120	13660
	12366	12536
	8920	11610
	8584	9346
	9170	11001
	10565	13382
	8277	13836
	8875	10662
	10092	9641
	8930	13319
	8842	11238
	9006	12035
	9618	11868

Stationary Agents		
Depth First	Breadth First	Flooding
	854	1081
	603	1125
	778	1170
	822	1096
	781	1103
	884	1186
	842	1191
	778	1167
	763	1156
	889	1096
	747	1185
	856	1148
	912	1128
	743	1157
	800	1191
	848	1154
	677	1098
	679	1164
	722	1158
	866	1175
	696	1148
	786	1167
	899	1190
	720	1161
	774	1098
	873	1145
	714	1167
	838	1159
	755	1163
	719	1145
	787	1149

Mobile Agents **Stationary Agents**