ADDIS ABABA UNIVERSITY
FACULTY OF INFORMATICS
DEPARTMENT OF COMPUTER SCIENCE

FAULT TOLERANCE MOBILE SOFTWARE AGENTS COMMUNICATION MECHANISM

By

ADDISALEM NEGASH

A THESIS SUBMITTED TO THE SCHOOL OF GRADUATE STUDIES
OF
ADDIS ABABA UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTERS OF SCIENCE IN COMPUTER SCIENCE

JULY 2004

Abstract

Mobile software agent systems have becomes an important tool for developing distributed applications. The use of mobile software agents is recommended in environment where network connection is not reliable and where there is low bandwidth. This is because mobile agents can perform the work with limited communication with the system that has created them. One important feature of mobile software agents is their ability to communicate to each other. Providing communication mechanism that guarantees delivery of message is that mobile software agent systems developer has to solve. A number of algorithms have been proposed to guarantee message delivery to highly mobile agents [13,8], but in our knowledge only one of them considers the existence of fault [8] and even that algorithm give responsibility transmitting message to a mailbox. In this thesis we have developed an algorithm that provides guaranteed delivery of message in the existence of fault particularly without assuming the location of mobile agent and without inhibiting movement of mobile software agent.

Acknowledgment

This thesis could not be completed without the help of many people. First and foremost, I would like to thank my advisor, Dr. Dawit Bekele, for his guidance and help. His support, his gentle way of teaching, his generosity and kindness, and his high standards of integrity have been inspirational to me and crucial to the completion of this thesis.

I would like to thank Dr. Yirsaw Ayalew, Dr. Solomon Atnafu, Mr. Jacob Eliosoff, Ato Girma Berhe and Ato Dereje Getaneh for their help.

All friends, colleagues and my families also play an important role in the accomplishment of the study and I thank them for their suggestions and advice. Most of all I have to thank my friends Ato Berhanu Tekle, W/t Helen Ayele, W/t Rahel Kifle and Ato Taddesse Tareke for their constant help. I would like to express my special thanks to my family. It is their love, understanding, and support that enabled me to accomplish this thesis.

I think that no list of acknowledgments can ever be complete, and this will not be an exception. So if you feel that I should have acknowledged you as well as the others, please feel acknowledged. I simply forgot to mention you.

Table of content

List Of Figures

1. Introduction

The emergence of computers significantly extends human being's computing ability, and the enhancement of communication technology extends computer's computing ability. The combination of computer and communication generates a whole range of computer network technologies, which change the traditional standalone computation on individual computer into distributed computation on connected cooperative computers.

A distributed system is attractive for creating cooperation between workers of an organization. Since organizations are after distributed to different locations, the employees need to share data. In addition, they also need to share resources. For sharing resources distributed system is the best solution.

Distributed systems are interesting for providing much higher processing power than a single computer can provide. For instance, calculating some astrophysics problem may need several of Pico Hz of processing power that even the most powerful computer cannot provide. Distributed systems organized as a grid of computer can easily provide this power.

In the traditional world of distributed systems, client/server model using remote procedure calls has been accepted as the best model to cope up with distributed. In addition, large applications have built using this technology. The conventional client server model has been adapted to object-oriented technology, giving rise to systems such as CORBA where objects are used as both client and server, and where RPC's are replaced by object invocations. Although both the traditional client/server model and the distributed object-oriented model perform well in a controlled, high-bandwidth office environment, they are not adapted to face challenges of slow and unreliable networks like Internet and Mobile environment.

The other technique used in distributed system to access recourses of other computers, is by using mobile code. In its simplest form, the concept of mobile code involves dynamically installing code on a remote host. In Web applications, applets and servlets are a common form of mobile code. The mobile code concept also appears in systems that extend the notion of remote procedure calls to transport the procedure to the server along with the call.

Many researchers extend the mobile code concept to remote evaluation, in which an object (code and data) moves from one host to another. The mobile agent abstraction extends this notion further by moving code, data, and state from one host to another. A mobile agent runs in one location, moves to another host, and continues at that host.

A mobile software agent usually differs from other forms of mobile code in that mobile agents have migration autonomy. By autonomy we mean that the agent can decide when and where to go.

Mobile agents offer many potential advantages over traditional approaches[19]. By moving the computation to host, which holds the data, it results to high throughput and low latency access to that data. Compared to more traditional client server approaches, mobile agents can avoid transmitting a large amount of data across the network, which is of particular value when the network is slow or unreliable. In addition to speed and reliability improvements, mobile agents can also help structure distributed applications. A service designed to relocate itself in the network to accommodate changing network conditions or the changing location of its clients can easily be written as a mobile agent.

Mobile software agents are suitable to develop distributed Internet application [19,31,32]. Since a mobile software agent is autonomous, it can tolerate the unreliable condition that may exist in the Internet. Distributed Internet applications are particularly important for developing countries like Ethiopia where it is too expensive By using application that use mobile software agents, it is possible to limit the effect of the unreliable Internet connection. Therefore developing countries like Ethiopia can benefit from Mobile software agent technology.

In last few years, several systems and programming environments have appeared to support the development of distributed applications based on mobile agents. Nevertheless there are still several open research issues to make the mobile agent technology widely appropriate. Among several technical problems that remain open, one is related to handling the agents' communication.

One of the basic properties of an agent is, its ability to communicate with other agents. At first glance, remote communication seems a contradiction with the goal of code mobility, i.e. to minimize communication over network. However, remote communication is helpful with conjunction with mobile software agent for monitoring mobile software agent's activity.

Many Mobile Software Agent systems seem to give a great emphasis on local communication, either using some sort of meeting abstraction, events for group communication or tuple space [13]. On the other hand remote communication is handled by conventional mechanisms like RPC, Java RMI and message passing, that either do not give guarantee for the delivery of messages, or enforce continuous connectivity with the message source, which in many cases defeats the very purpose of using mobile agents [13]. The attachment of message and message source becomes more problematic in the context of unstable network connection [13].

This study deals with communication mechanism of mobile software agent, which provides guaranteed message delivery in the existence of fault. In this study, only disconnection is considered as fault. However we believe that it can be easily generalized to consider other types of faults.

This thesis is organized as follows:

- Chapter 2, PROBLEM STATEMENT AND METHODOLOGY, presents the problem to be solved and the methods that are employed for the research.
- Chapter 3, BACKGROUND, gives background information regarding mobile software agents and the communication used by current mobile software agent system.
- Chapter 4, RELATED WORK, reviews some of the related works done in the area of message delivery to mobile software agent: - these include Murphy's Algorithm, Mailbox based message delivery, and so on.
- Chapter 5, MESSAGE DELIVERY TO HIGHLY MOBILE SOFTWARE AGENT, presents an algorithm that provides a guaranteed message delivery to a highly mobile software agent by analyzing and extending Murphy's Algorithm.

- Chapter 6, IMPLEMENTATION, presents a prototype developed in order to demonstrate the communication algorithm presented in chapter 5 of this thesis.
- Chapter 7, CONCLUSIONS, highlights some concluding remarks and outcomes from the research.

2. Problem Statement and Methodology

2.1. Problem Statement

The problem we are interested in is the delivery of messages between two mobile software agents in the existence of fault. Amy L. Murphy has studied the problem of communication between mobile software agents and develops an algorithm to provide guaranteed delivery of message [13]. Murphy's algorithm considers that the communication medium is reliable. However mobile software agent is recommended in the network where network connection is not reliable. We will try to address this problem in the existence of fault by focusing on disconnection.

In various situations, mobile agents need to communicate with each other but the free movement of mobile agents makes determining the location of mobile software agent a difficult task and makes in delivering of messages to mobile agent a difficult problem [8,12,13,14,15,16,32].

Many Mobile Software Agent systems seem to give a great emphasis on local communication, either using some sort of meeting abstraction, events for group communication or tuple space [13,18,22]. Remote communications is handled by conventional mechanisms like RPC, Java RMI and message passing, that either do not give guarantee to the delivery of messages, or enforce continuous connectivity with the message source, which in many cases defeats the very purpose of using mobile agents [13]. The attachment of message and message source becomes more problematic in the context of unstable network connection [11]. Since in such kind of network it is possible that the connection between the node containing the message and message source fail, either the application fails because of this or it should be slowed down by waiting until the two nodes are reconnected.

To utilize the potentials of mobile software agents, it is important to design a communication infrastructure that guarantees the delivery of messages without modifying the nature of Mobile Software Agent. That is communication mechanize that doesn't assume mobile software agent location and inhibit mobile software agent's movement is needed. Therefore, this thesis deals on finding communication mechanism which provide guaranteed message delivery in the presence of fault with out inhibiting mobile agent movement and without assuming the location of mobile software agent.

2.2. Assumptions

In this study the following assumption is made:

- A network that may have temporary failure
- The network will not crash while agent is moving

2.3. Delimitations and limitation

This study is delimited to investigate communication mechanisms for Mobile Software Agents that insure delivery of message in existence of unreliable network connection. Time and cost constraints have limited this study to the indicated issue.

2.4. OBJECTIVES

General objectives

The objective of the study is to investigate the mobile software agent communication mechanism that insures delivery of messages without changing the nature of Mobile Software Agent in the presence of fault.

Specific objectives

The specific objectives of this study are to investigate mobile software agents communication mechanisms that

- can tolerate disconnections.
- allow asynchronous message communication.
- guarantee reliable delivery of messages
- Don’t assume the location of the gent
- Don’t inhibit agents movement

2.5. Methodology Used

The methodology used in the study is the following:

- Different related researches and background theory have been revised through literature review.
- An algorithm that can be adapted to provide the desired communication mechanism has been selected and it has been adapted to work in the existence of disconnection.
- The prototype that is used to demonstrate communication mechanism implemented by the algorithm, has been developed and tested.

3. Background

3.1. Overview Of Mobile Software Agent

A mobile agent is a program, which represents a user in a network and is capable of migrating autonomously from one node to another node, performing computations on behalf of the user [7]. In order to support the agent mobility, the infrastructure should provide a language-level primitive that an agent can call to move itself to another node.

An agent is able to cooperate with other agents in order to perform complex and dynamic tasks. It is also capable of identifying and using resources specific to any node on which it finds itself.

The mobile agent paradigm differs from the traditional client/server approach in the following ways [31,32]. In the client-server paradigm, resource owners (servers) are physically distant from their clients (users). The communication among these parts (client and server) occurs through a network of computers, being mediated by mechanisms such as remote procedure calls, message exchange, sockets and so on. In client-server paradigm, the reliability of the communication links and the synchronicity of the remote procedure calls are important requirements of the majority of such applications. In contrast, in the mobile agent paradigm, the agents migrate to interact locally, at the same host as the resources. By moving from one location to another, agents can dynamically change the interaction quality, reducing the communication cost and the overhead added to maintain the reliability of communication [5]. In short, the mobile agent paradigm differs from client-server paradigm since in mobile software paradigm; code is moved to where data is stored rather than moving data is moved where the code is stored. Moving code to data is more reliable than moving data to code in low bandwidth and unreliable network like Internet. This is because the following facts.

- The intermediate data shouldn't be moved; as a result it saves bandwidth
- The applications are only vulnerable to network disconnection during the agent transfer, not during the interaction with the resource.

Mobile agents differ from other mobile code or program in that they provide autonomy, that is, they themselves can decide dynamically where and when to travel to a particular destination [9].

3.2. Mobile Agent Architecture

A mobile agent is composed of three parts:

- The agent code part which corresponds to a certain algorithm,
- The agent execution thread (with an execution stack) and
- The agent data part that is corresponds to the value of the agent's global variable.

Each agent runs independently of each others, is self-contained from a programmatic perspective, and preserves all of its state when it moves from one network node to another.

The mobile software agent travels between execution environments called places. A place is a control within an agent system that provides a uniform environment in which an agent can execute [2]. It provides an infrastructure for managing mobile agents, enforcing security policies and accessing local resources.

Places have the following actions

- Facilitate agent execution,
- Support agent communication with other agent and with the server and
- Protect themselves and agents from attack

Mobile software agent system usually provide one or more of the following services:

- Code Transfer: Typically, the code to be executed by an agent is not available on all of the agent's destination hosts. Thus, most agent systems provide a means for transferring the code of the agent either by transferring it from a centralized location (code base) or from the host, where the agent was previously executed.
- Agent Identifier: Many agent systems assign a unique identifier to agents so that they can address each other and to register themselves at a naming server if necessary.

3.3. Communication Model of Mobile Software Agent

One of the basic abilities of an agent is its ability to communicate with other agents[12]. During its nomadic life, an agent needs to communicate with other entities. For example, communication is needed to control the child agent, to coordinate agents' activities with each other or to access the resources of the network.

Communication mechanisms of mobile software agent can be seen along with two orthogonal dimensions: the number of mobile software agents involved in communication and their respective location [34]. Hence, the first dimension partitions the space of mechanism into point-to-point and multi-point mechanisms while the second distinguished among local and remote communication.

Point-to-point mechanisms enable communication between two mobile agents. The more primitive mechanism is message passing, although no existing mobile software agents actually use it for communication between agents moving arbitrarily. Usually, some form of remote procedure call (RPC) is provided instead. This may involve invocation of bare procedure or method invocation on remote object.

Multi-point mechanisms enable communication between three or more mobile software agents at a time. Shared memory is by far the mechanism most frequently used to achieve multi-point communication. Most of the existing mobile agent uses this method for local communication, although it could be extended for remote communication [18].

Communication mechanisms of mobile agent can also be divided into two groups [34]: direct communication, where the communicating peers explicitly identify themselves;

and anonymous communication, where the sender does not know the identity of the recipients. Direct communication can involve either only two peers or a sender and a set of receivers (multicast). Forms of anonymous communication being supported are events, spaces and tuple spaces.

In direct coordination models, agents start a communication by explicitly naming the partners involved. In the case of inter-agent coordination, two agents must agree on a communication protocol, typically peer-to-peer. Access to local resources generally uses client-server coordination, since a hosting environment usually provides local servers for its resource management.

Most Java-based agent systems—like Sumatra, Aglet and Odyssey-adopt the client-server style typical of object systems, and can also exploit low-level message passing via TCP/IP [7,33,34]. Agent Tcl [18] provides direct communication between two agents, based on message passing, and also provides asynchronous communication modes. Thus, they still rely on direct communications based on remote procedure calls for their services. Although middleware systems can facilitate the use of independently developed components, using them to enable direct mobile-agent coordination will cause latency and reliability problems [12].

In meeting oriented communication model, an agent can interact with no need of explicit naming of the involved partner [12,34]. Agents join either explicitly or implicitly known meeting points; afterward, they can communicate and synchronize with the other agents that participate in such meetings. Ever-open meetings abstract the role of servers in an execution environment; application agents can open further meeting points as needed. To avoid the problems related to non-local communication, such as unpredictable delay and unreliability, meetings often take place in a given execution environment, which allows only local agents to participate.

The Ara mobile-agent system implements a typical example of meeting-oriented communication model [34]. The concept of event-based communication and synchronization, defined by the Object Management Group and implemented in the Mole mobile-agent system offers a sophisticated form of meeting-oriented communication[36]. Specific synchronization objects, which agents must share the reference to, assume the role of meetings. Accessing one of these synchronization objects allows agents to implicitly join the meeting.

Meeting-oriented communication models partially solve the problem of exactly identifying the partners involved, although they cannot achieve the anonymity of full spatial uncoupling: Agents must share at least the common knowledge of the meeting names. The major drawbacks of the models derive from there enforcing synchronization among interacting agents[12,34]. Since the schedule and the position of agents cannot be predicted in many applications, the models run a high risk of missing interactions. Furthermore, if the meetings are not locally constrained, they must be implemented by message passing, thus inheriting the efficiency and reliability problems of direct coordination models.

In blackbox-based model, agents interact via shared data spaces [12,34]. The data are used as common repositories to store and retrieve messages. In this sense, interactions are fully temporally uncoupled, but, because agents must agree on a common message identifier to communicate and exchange data via a blackboard, they remain spatially coupled. To overcome scalability problems, a local blackboard can be associated with each hosting environment.

Several systems propose and implement blackboard- based communication models for mobile-agent applications [34]. In Ambit, a formal model for mobile computations, agents can attach messages to a blackboard on a given site; another agent can retrieve and read these messages when arriving at the same site.

Linda-like coordination model [35], uses local tuple spaces as message containers similar to blackboards. In addition, a tuple space bases its access on associative mechanisms. The system organizes information in tuples and retrieves it using associative pattern matching. This approach enforces full uncoupling, requiring neither temporal nor spatial agreement.

4. Related work

In this chapter we will discussed algorithms developed to deliver messages for mobile agents. Since the characteristics of mobile software agents are similar to mobile devices, in this chapter, those works done related to delivery of messages to mobile devices will be included as well.

Basically, the complexity of message delivery problem in Mobile Software Agent System or in mobile computing environment in general is caused by the migration of the actors (the agent or the mobile unit) [8]. Various approaches like central server, forwarding pointer, broadcasting, hierarchical location directory have been proposed to overcome the message loss caused by migration of recipients.

4.1. Approach used to deliver messages to mobile agents\devices

Central server

There are three types of central servers called central forwarding server, central query server and home server. Central forwarding server and central query server hold the current address of all agents in the network. However home server only holds the current address of some agents, usually the agents created on that server.

In central forwarding server approach, before an agent migrates, it should inform the server about its migration and when the agent arrives, it should tell the server that it has finished its migration. If agent wants to send message to another agent, it sends the message to central server and the central server forward the message to the agent to which the message is sent.

Central query server approach is similar to the central forwarding approach. However, in central query server; the server will give an address of the agent (to which the message will be send to) to the agent which wants to send a message, instead of forwarding the message.

Home server approach works in the same way as central Forwarding approach except that few of agents' address are stored. Meaning every agent has a home server and agent's name contains its home address. Whenever an agent moves at some node in the network, the agent registers its current address in its home server. When an agent wants to communicate with another agent, it sends a message to the home server of the agent to which the message is sent. Upon accepting a request, the home server forwards the message to that agent to which the message is sent.

Central server approach will be a bottleneck for mobile environment particularly for highly mobile once. This is because informing central server will be an overhead and also disconnection of these servers will disturb the agent's performance.

Forwarding Pointer

In the forwarding of pointer scheme, before an agent migrates, it leaves a pointer which points to the target host in its current location. When a message is sent to an obsolete address of the recipient, the message is routed along the forwarding pointer.

The disadvantages of using forwarding pointers are:

- If one host in the sequence crashes, it is difficult to locate the agent
- The link way becomes too long, which results in loss of performance.
- If the recipient migrates frequently, the message may keep chasing the recipient and cannot be received until the death of the recipient.

Broadcasting

A simple broadcast scheme assumes a spanning tree of nodes through which a message sent by any node. This node then broadcast the message to it neighbors, which broadcast the message to their neighbors, and so on until the leaf nodes are reached. This, however, doesn’t guarantee delivery of message. This is because an agent traveling in the reverse direction with respect to the propagation of the message may not be able to get the message. If the agent is being transferred at same instance the message is propagating in the other direction, the agent and the message will cross in the channel and message delivery will never occur.

There are other types of broadcast approach called query broadcast and notification broadcast. In query broadcast, when an agent wants to send a message to another agent, the host sends a query asking an address of that agent to all nodes. When the host gets the address, the message will be send by using that address. In notification broadcast, whenever an agent reached at a site, it will broadcast its current address to all nodes of the network. In this approach all nodes will have a guess address of all agent and use that address to send message to an agent. When a node receives a message to an agent which is not in the node, it keeps the message until the new location of the agent is broadcasted and then the host forwards the message to that location.

The cost of communication of query broadcast seems better than the other broadcasting approach for large messages. However, this is not appropriate for highly mobile agents since their address will be changed frequently. For the same reason, notification approach is not appropriate for highly mobile agent.

The simple broadcast schemes have less reliance on the agent home for, agent tracking or message forwarding, thus it can used to deliver messages for highly mobile agents. It can be implemented in local Internet domain or local Ethernet. However, message in transit make the broadcasting approach to not provide guaranteed delivery of message.

Hierarchical Schemes

In the hierarchical schemes, a tree like hierarch of servers forms a location directory (similar to DNS). Each server in the directory maintains a current guess about the site of some agents. Site belong to regions, each region corresponds to a sub-tree in the directory (in the extreme cases the sub-tree is simply a leaf-server for the smallest region, or the whole tree for the entire network). Each region corresponds to a sub-tree in the directory. For each agent there is a unique path of forwarding pointers that starts from the root and ends at the leaf that knows the actual address of the agent. Messages to agents are forwarded along this path.

The hierarchical scheme scales better than forwarding pointers and central servers. It supports locality of mobile object migration and communication. However, the hierarchy is not always easy to construct, especially in the Internet environment. The hierarchical scheme itself cannot guarantee message delivery since messages might also chase their recipients under this scheme.

4.2. Guaranteed message delivery

In the previous section, we discussed about various approaches for delivering messages for mobile agents. None of them provide guaranteed delivery of message by themselves to highly mobile software agents.

In our knowledge, there are only few algorithm designed to guarantee delivery of message to mobile software agents. One of the algorithms is designed by Murphy [13], which uses ideas of distributed snapshot and diffusion computation to guarantee delivery of message for highly mobile agent in the absence of fault. There is also another algorithm designed by Cao, Feng, Lu, and Das [8], which tries to attach mobile agents to mailbox and achieve delivery of message by using the mailbox.

4.2.1. Murphy's algorithm's

Amy L. Murphy proposed algorithms which uses technique similar to distributed snapshot to guarantee the delivery of message in physical as well as logical mobile environment in the absence of fault. Physical mobility deals with the mobility of physical components and logical mobility deals with mobility of mobile software. One of the algorithms is based on the concept of diffusion computation proposed by Scholten and Dijkstra[27] and the others are based on Distributed snapshot.

4.2.1.1. **Using Distributed Snapshot for Physical mobility**

Murphy adapts the snapshot algorithm to message delivery. In order to avoid confusion in terminology between the control traffic generated by the snapshot algorithms and the data traffic containing the information being communicated to the mobile unit, they use the term announcement to refer specifically to the data message being delivered while a message can be either data or control (including mobile unit)

The snapshot algorithms were developed to detect stable properties such as termination or deadlock by creating and analyzing a consistent view of the distributed state. The snapshot algorithm is used to record the state of the message in the network graph having FIFO channels.

The snapshot algorithm consists of two main localized actions to collect the local snapshot: the processing of the control messages (markers) and the arrival of the messages to be recorded. The marker arrival rule states that when a marker arrives at a node not involved in a snapshot, the node begins its local snapshot by recording the processor state, and then sends the marker on all outgoing channels.

The message arrival rule of the snapshot states that if the message arrives at a node from channel C before the marker arrives on channel C, and the node is in the middle of the local snapshot, the message is to be recorded as on the channel during the snapshot The local snapshot is complete when the marker has arrived from all incoming channels

Murphy adapts the snapshot algorithm to perform message delivery in the dynamic, mobile environment. In the algorithm mobile unit is considered as persistent messages (the mobile unit is assumed to be somewhere in the network) and the message to be delivered are considered as marker.

Since the mobile unit is considered as a message, and the snapshot records the location of messages; the global snapshot of the mobile system will show the location of the mobile unit. Therefore, one option is to simply deliver the announcement directly to this location; however, it is possible (and likely in systems with rapidly moving mobile units) that the mobile unit will move between the time its position is recorded and when the announcement arrives at the recorded position. Therefore, they alter the snapshot recording to delivery of the message by augmenting the control messages with the announcement and changing the recording of messages into the delivering of announcements.

4.2.1.2. Using Distributed Snapshot Logical Mobility

As the one done for physical mobility, this algorithm is also based on distributed snapshot algorithm (The Chandy-Lamport algorithm). Here also mobile unit are considered as messages and the actual message assumed as markers.

This algorithm differs from the one for physical mobility in that here, the algorithm allowed for concurrent delivery of message. Concurrent message delivery is done in the way that message delivered before the end of the delivery of another message will be stored in the buffer of the node and when the snapshot of the previous message at the node is cleared, the message stored in the buffer is assumed as newly arrived one.

This algorithm has also version, which allow the graph to grow as the mobile agent moves. This version has also another responsibility of constructing the graph while the agent moves.

4.2.1.3. Based on diffusing Computation

Diffusion computation has been used by Amy L. Murphy to develop a guaranteed message delivery [13]. By equating the root node of the computation to the concept of a home agent from Mobil IP and by replacing the messages of the computation with mobile units, the result is an algorithm which instead of tracking a computation as messages are passed through a system of processing nodes, tracks a movement of mobile unit as it visit various base stations in the system. Essentially, the graph of the Dijkstra-Scholten algorithm defines a region within which the mobile unit is always located. Although this is not directly a message delivery algorithm, by propagating the data message to be delivered throughout this region, message delivery can be achieved.

Diffusing computations have the property that the computation initiates at a single root node, while all other nodes are idle. The computation spreads as messages are sent from active nodes. The basic idea of the algorithm is maintaining a spanning tree that includes all active nodes.

This diffusion computation helps to track the mobile unit. Here they consider the mobile unit in place of message. When a mobile unit is sent from an active node to an idle node, the idle node will added as a child of the former in the tree. Mobile unit migrates among tree nodes have no effect on the structure but may activate idle nodes still in the tree. An idle leaf node can leave the tree at any time by notifying its parent. Termination is detected when an idle root that remains in the tree.

Since the sender of the message cannot know whether or not the destination node already in the tree, maintaining the tree from parent to child is difficult. As the result, the tree for message delivery is maintained from child to parent.

For message delivery is assume that the message (the data message to be delivered) originates at the root and it is relied on the property that there is always a path from the root to the mobile unit along edges in the tree. It is only necessary to send the message along edges in the spanning tree. But, because this tree is maintained with pointers from child to parent, the message must be propagated along the successor edges, from parent to child. When a message arrives from a source other than the parent, the message is rejected. In this manner, the message is only processed along the tree paths

To cope up with the migration of mobile unit, each node stores a copy of the message until delivery is complete or the node is removed from the tree. Storing the message in this manner ensures that the mobile unit cannot leave a region without receiving a copy of the message.

4.2.2. Mailbox based message delivery

In this approach, each mobile agent has mailbox. When an agent wants to communicate with another agent, it sends message to that agent's mailbox. An agent can get a message from the mailbox by using push and pull technique. In the push technique the mailbox sends the message to mobile agent. Here the mobile agent must send its address to its mailbox and the mailbox will forward any message send to the agent. In pull technique, the mobile agent keeps the mailbox address and retrieves messages from it.

In the mailbox-based scheme, the mailbox is detached from its owner agent. Before migration the agent can decide to take its mailbox with it. It is possible to make mailbox stationary, to migrate with owner agent or to follow the owner agent with some distance.

To deliver message to mailbox, home-server based, forwarding pointer or distributed registration protocol can be used. In home-server protocol, the mailbox is stationary in the home server of the owner agent and message is sent to that server. In the forwarding pointer protocol, before a mailbox migrates, it leaves a pointer which points to the target host, in its current location. When a message is sent to an obsolete address of the mailbox, the message is routed along the forwarding pointer. In distributed registration based protocol, before the mailbox migrates, it deregisters itself from all hosts visited by it and wait an acknowledgment from them. After arriving to the new host, it register the new address to all hosted previously visited by it.

Guaranteed delivery of message can be provided by the protocol used to track mobile agent and mailbox. Distributed protocol can be used to guarantee delivery of message to the mailbox. Synchronous message communication between the mailbox and owner agent can guarantee delivery of message from mailbox to agent.

Mailbox based message delivery can guarantee message delivery but will be bottleneck for highly mobile agent since an agent should synchronize with mailbox. Since mobile agent system should manage both the mailbox and the mobile agent, the implementation this system is also complicated.

5. Message Delivery To Highly Mobile Software Agent

5.1. Introduction

The objective of this chapter is to present an algorithm that provides guaranteed message delivery to a highly mobile agent in the existence of fault. We only consider disconnection as a fault; however we believe that the algorithm can be easily modified to tolerate other kinds of faults.

The algorithm adapts the Algorithm developed by A. Murphy [13], which provides guaranteed delivery in the absence of fault. Our algorithm modifies the Murphy's algorithm to work in the existence of disconnection.

Our algorithm provides guaranteed delivery of message in static and dynamic network graphs. In the case of static network graph, it is assumed that the graph is pre-established before the algorithm runs and the algorithm doesn't worry about the construction of the network graph. In the case of dynamic network graph, the graph should be constructed dynamically and the algorithm has additional responsibility of constructing the network graph dynamically while the agent moves.

In section (5.2) we will discuss Murphy's algorithm on which our algorithm is based on, and in sections (5.3) and (5.4), we will discuss the algorithm that we designed in the study. In the last section, we will summarize the chapter and will discuss how the algorithm can be modified to work in the existence of faults other than disconnection.

5.2. Overview of Murphy's algorithm

In this section we will discuss an algorithm, which is developed by Amy L. Murphy [13], which provides guaranteed message delivery in the absence of fault to a highly mobile software agents. Murphy's basic algorithm deals with a single message delivery in a static (predefined) network and its subsequent enhancements that allow multiple messages delivery and construction of the network graph dynamically while the agent moves.

The logical model used in Murphy's algorithm is a typical network graph of nodes and channels, where the nodes represent the server willing to host the agents, and the channels represent links between nodes that can be used for transporting messages as well as agents. These channels are directional and FIFO.

The main objective of the algorithm is to guaranty the delivery of message to highly mobile agents. Simple broadcasting or forwarding schemes, recommended by some authors may create situations where the message never reaches the agent. This happens when the broadcasted messages and the agent travels in reverse direction in the case of broadcast scheme (figure 5.2), and when the agent moves as fast as the message in the case of forwarding scheme (figure 5.1).

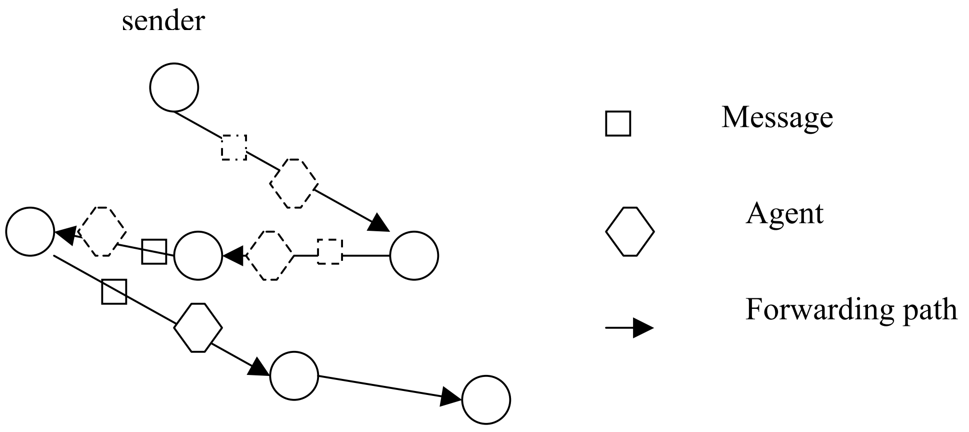


Figure 5-1 Forwarding pointer scheme

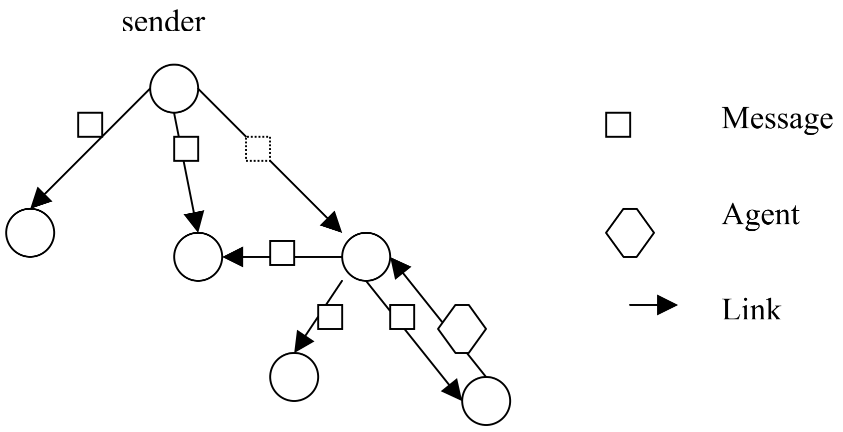


Figure 5-2 Broadcasting scheme

5.2.1. Basic Murphy's algorithm

The basic algorithm considers that channels are predetermined and assumes only a single message is transmitted at a time. The main ideas of the algorithm are to broadcast the message through the network as a broadcast scheme and, to store the message at each node, so that the agent that arrives at the node from a direction opposite to the broadcast can get the message (figure 5.3)

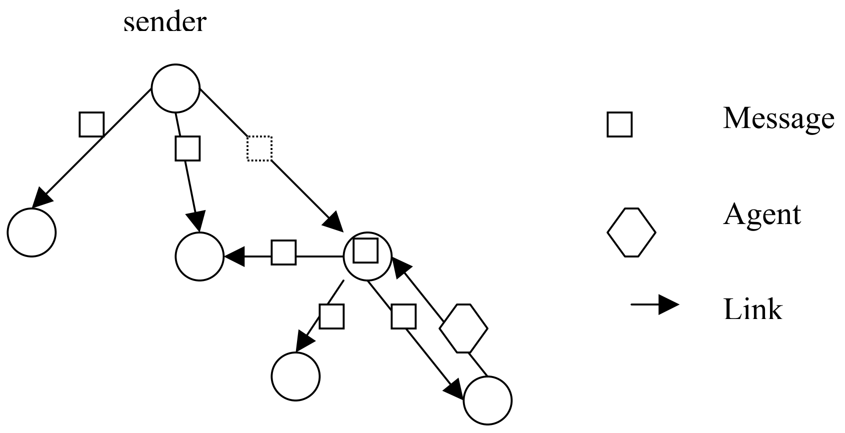


Figure 5-3 Broadcasting and coping messages in each node

In figure 5.3, the agent reaches the node after the message has passed. However, unlike the simple broadcast scheme, since the node keeps a copy of the message, the agent will find the message at the destination node even after the wave of the broadcast has passed.

A major problem here is to know for how long the node keeps a copy of the message, because it is not realistic to keep the copy of the message indefinitely. To determine the length of time that a message is kept at a given node, Murphy uses a technique based on the technique used for snapshot by Chandy and Lamport[1].

The main goal of the technique is to insure that all agents in transit on incoming channels while the message is broadcasted on outgoing channels get the message before it is deleted, in order to avoid problems such as the one shown in Figure 5.2. For this purpose, at each node, a state that can be FLUSHED or OPEN is associated with each incoming channel. Initially all channels are OPEN. When a message arrives by a channel the state of the channel is changed to FLUSHED and the message is stored locally. At this point, the situation where the message and the agent miss each other while in transit (figure 5.2) cannot occur since the message will be copied at a node before broadcasted.

The copy of the message is kept on the source node until it is sure that all agents in transit and in opposite direction with the message broadcast have reached the node, i.e. until all incoming channels have the state FLUSHED. All incoming channels of a node are FLUSHED means that the message has arrived from all incoming channels. Therefore, since the channel is FIFO, any agent that comes from these channels has already obtained the message from it source node. For example, in figure 5.4 Agent 1 and Agent 2 should necessarily receive the message at Node 2 and Node 3 respectively this is because, the agents received the message prior to the moment the agents have left Node 1. It is therefore, no more necessary to keep the copy of the message at Node 2 and Node 3.

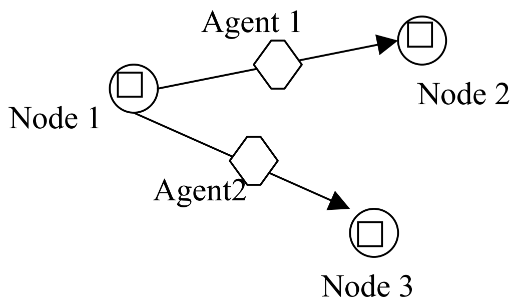


Figure 5-4 The effect of FLUSHED incoming channels

5.2.2. The Enhanced Algorithm for Multiple message delivery

The Murphy's basic algorithm enables delivery of a single message. However, it is important to have the possibility to deliver multiple messages. The Murphy's basic algorithm can be used to deliver more than one message. However it requires checking the termination of delivery of one message before starting to deliver another message. In distributed systems, checking termination of an algorithm is complex and time consuming. Therefore allowing concurrent message delivery without the need of detecting termination of an algorithm is needed.

Murphy extended the basic algorithm to provide multiple and concurrent message delivery. This algorithm works for messages sent by the same source. However it can be easily generalized to deliver messages sent from several sources.

The main goal of the enhanced algorithm is to insure the delivery of messages by keeping only one message active at a time. For this purpose, the channels can have another state called BUFFERED (figure 5.5). As in the case of single message delivery the channels have OPEN state initially. When a channel transports a message, that message is considered to be the active message and the state of the channel becomes FLUSHED. The node give the active message to all agents hosted and broadcasts the message in all outgoing channel. When an agent arrives at the node, it will be given the active message. When a channel that is FLUSHED delivers a message, the message will be stored in a buffer queue and the state of the channel will be changed to BUFFERED.

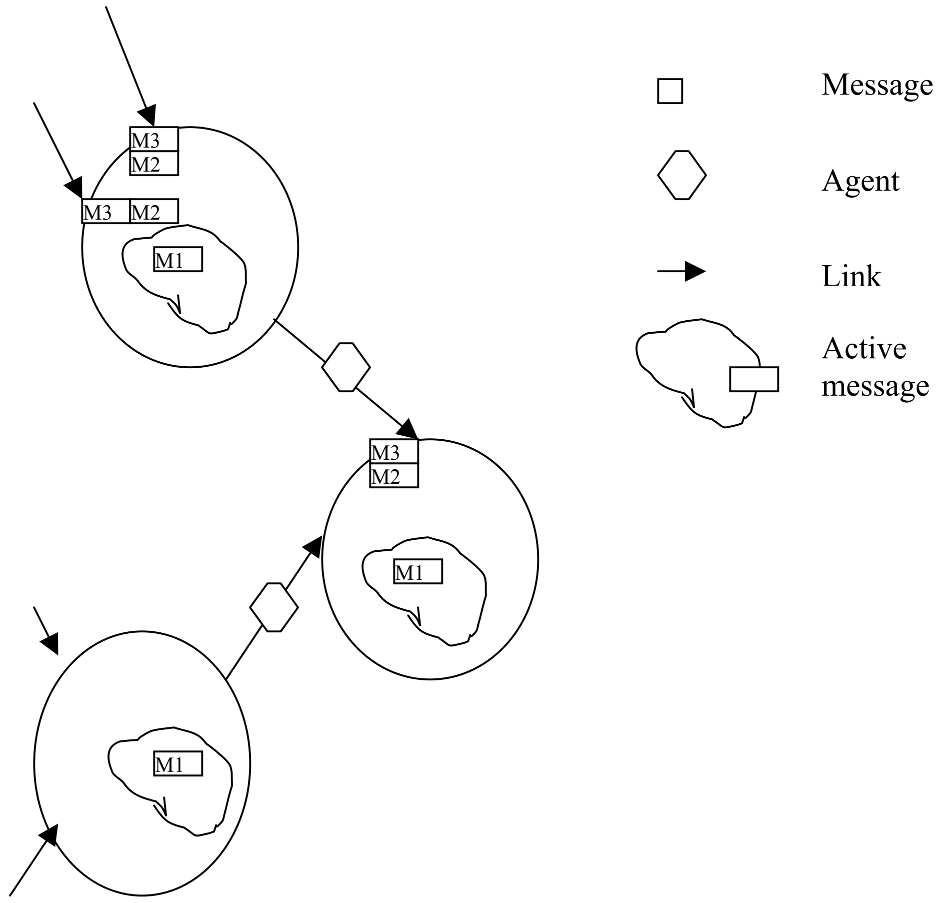


Figure 5-5 Multiple message delivery

The messages in the buffer queue remain there until the active message is removed. The active message is removed when all channels becomes FLUSHED or BUFFERED, since storing the active message is no more necessary after that. At this point, Murphy's algorithm clears the active message and makes the state of all channels OPEN. Messages in buffer queues are then considered as newly arrived. Here, changing the channels having BUFFERED channels to OPEN will not create a problem since the messages arrived by it will be considered as new..

5.2.3. The Enhanced Algorithm for Dynamic Network

Up to now the discussion considers that channels are pre-established. However Murphy has also designed an algorithm for dynamic network graph, which is constructed dynamically, while the agent moves. This algorithm has one additional responsibility that is network graph construction. The algorithm is done for agents and messages of the same sources. However, running the algorithm concurrently can provide guaranteed message delivery for agents and messages of different sources.

The network graph is constructed as follows. Initially only the node on which an agent is created is considered as active. When an agent moves from an active node to another node, which is not active, the node that the agent arrived at and channel that the agent has been transported become active.

This algorithm works as the one for static graph but sometimes the agent may be forced to stay at a node until some messages arrive or delay the activation of construction of an incoming channel of a node until some messages are arrived and cleared from the node. This is done to handle difficulties caused by the free movement of mobile agents and the fact that the graph is constructed by considering all agents as explained below.

One of the difficulties caused by the free movement of mobile agent is called destination ahead of source. This difficulty occurs when mobile agents migrate to a node which have incoming channels constructed by other agents. To discuss the difficulty by using an example, we assume that there are three nodes called X, Y and Z in the network and X is active initially. When X sends an agent a1 to Y, both Y and channel connecting X and Y becomes active. Assume that X sends message 1 and 2 and then sends another agent called a2 to Z. At this time, node Z and the channel connecting X and Z will be active. If a1 leaves Y before message 1 and 2 are arrived, a situation called destination ahead of source will occur (figure 5.6). At this point, if the messages are forwarded blindly, message order is possibly lost and messages can possibly keep propagating in the network without ever being deleted. The solution they use to solve this difficulty is to hold the agent at Z until the message 1 and 2 is received and, when these messages are arrived, the messages will be deleted after they are delivered to the detained agent (a1). This action inhibits the movement of agent but Murphy argues that this is taken place only for a time proportional to a diameter of the network.

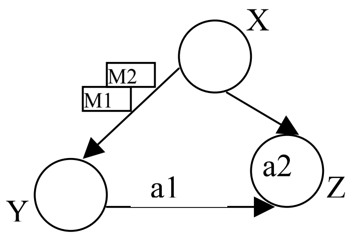


Figure 5-6   Destination ahead of source

There is also other potential problem called source ahead destination. This is caused when a new incoming channel has been constructed while the node or the old incoming channels of the node holds messages To discuss this difficulty, we use the above example except instead of X sends another agent to Z, assumes that Z sends an agent to Y. Here a situation in which Y waits the channel coming from Z to be FLUSHED, before deleting message 1 and starting to process message 2. However, the channel coming from Z cannot transport message 1(figure 5.7). This situation is called source ahead of destination. To solve source ahead of destination problem, the activation of the channel connecting Z with Y will be delayed until Y catch up with Z. In this example, the channel will be delayed until message 2 is processed.

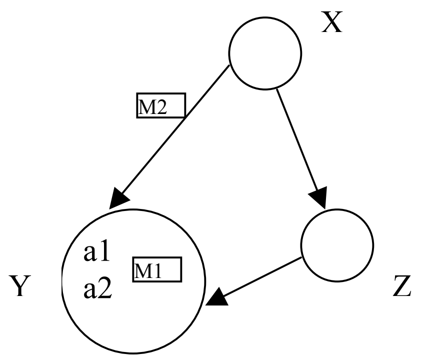


Figure 5-7 Source ahead of destination.

5.3. Fault tolerant communication algorithm

In this section, we are going to modify Murphy's algorithm which is presented in the previous section to work in the existence of disconnection. In section 5.3.1, we will present the logical model used in the algorithm and in section 5.3.2 and 5.3.3, we will present our algorithm for static network. In the last section, we will present our algorithm for dynamic network.

5.3.1. Model

The logical model (architecture) that is used for the algorithm is a typical network graph of nodes and directed channels. The nodes in the logical model represent servers willing to host agents and channels that represent links used to connect nodes and that can transport both agents and messages. In the algorithm, the channels are assumed to be FIFO and directional (figure 5.8).

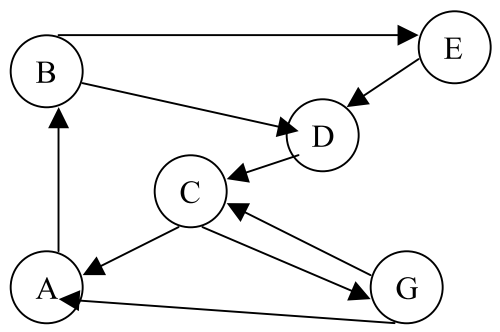


Figure 5-8 Logical model for the network graph

The algorithm considers both static and dynamic network graphs. In static network graph, the channels are assumed to be established before the algorithm starts in contrast in dynamic network graph, channels will be constructed while the agent moves.

In static network graph, we assume that there is always a path between two pair of nodes. This doesn't mean that a full connection should exist but a node should be reachable from any other node. For example in figure 5.8, there is no direct link between node B and Node A but node B can get node A through node D and node C.

The algorithm assumes that agents have global name, which identify them uniquely, and messages have IDs, which are always increasing. It is assumed that the agent name holds the name of its creator node or home node.

We also assume that, it is possible to detect disconnection as well as reconnection. It is also assumed that, it is possible to know whether the message which is sent, is delivered at the destination or not.

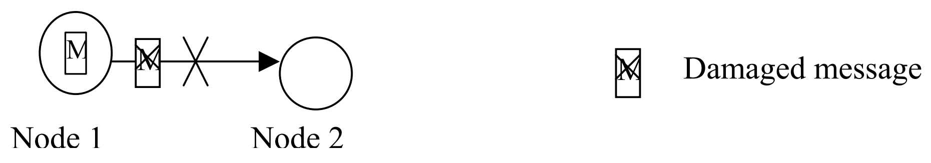
5.3.2. Delivery of Message with Static Network Graph

We start describing our algorithm with the simplest case where the network graph is static that is it is fully constructed prior to delivery of message and one message is assumed in the network. We also discuss the algorithm to deliver multiple messages.

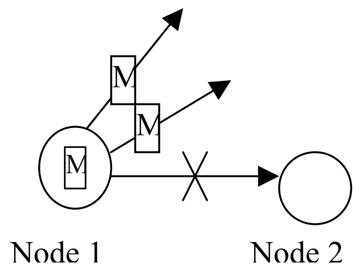
As discussed in the problem statement, the main objective of this study is to provide an algorithm that delivers message to a highly mobile software agent even in the existence of fault.

We adapt Murphy's algorithm discussed in the previous section that does not work with the existence of fault. We believe that making assumption on the absence of fault specially disconnection is not appropriate, since mobile software agent usually recommended in the area where network connection is not reliable.

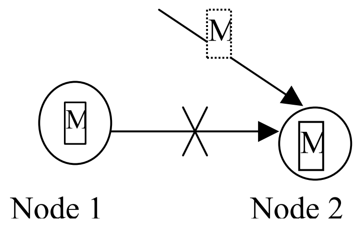
Assume that Murphy's algorithm is applied in the network where the connection is not reliable. When a channel is disconnected the message in the channel will be lost, and the node to which the channel is an outgoing one can't send the message through the channel (Figure 5.9a, b). In addition, some node may not be able to clear the copy of the message stored at node, if one of its incoming channel is disconnected (Figure 5.9c). It seems that transportation layer can handle this but this depends on time required to reconnect the channel. For example, if the network is established using TCP, their established TCP connections can abort during periods of disconnection and many TCP implementations default to timeout values of a few minutes [28]. Therefore Murphy's algorithm can't provide guaranteed delivery of message in the existence of disconnection particularly for extended disconnection.



a. Message in transit damaged by disconnection



b Node is unable to send message through disconnection channel



a. Node may be unable to clear the messages if one of its incoming channels is disconnected

Figure 5-9 The effect of disconnection

We extend Murphy's algorithm to work in the existence of disconnection by changing the way of message broadcasting. When a message is broadcasted in every channel, the node identifies the channel for which the channel is disconnected while transporting the message and resend the message when they are reconnected. The broadcasting of message will be finished when all channels transports the message. Here we are sure that *message in the node will not be cleared before the successor of the node have it, since broadcasting of message is finished when all channels transports the message.*

5.3.3. Multiple Message Delivery

Since Murphy extends the algorithm that she designed for single message delivery to provide guaranteed delivery of multiple message, the algorithm does not provide guaranteed message delivery in the existence of disconnection. In this section we extended Murphy's multiple messages algorithm to work in the existence of disconnection. As in the case of Murphy's algorithm, our algorithm is limited for messages of a single source. However running this algorithm concurrently will provide guaranteed delivery of message of any source.

As in the case of single message delivery, disconnection can be handled by changing the way of message broadcasting. That is, if some of the channel is disconnected during broadcasting, the message will be sent after the channel is reconnected and message broadcasting will be end after the message sent by all channels. However, delaying the node work until the successor is reconnected is not reasonable, since other messages may be waiting in the buffer queue. In addition the failure of message broadcasting in channels don't have an effect on the node which broadcasts the message but have effect on the successor of the node.

We modify our algorithm in the way that it doesn't affect the work of a node as follows. The node broadcasts the message to all connected outgoing channels and then identifies the disconnected channels before broadcasting and while broadcasting. If there are disconnected channels, the node stores the message broadcasted with the channel ID in the temporary store. The broadcasting is finished after sending the message in all connected channels and recording the message with IDs of disconnected channels.

When reconnection of an outgoing channel is sensed, a node sends the messages which are stored in temporary store with the ID of the channel. After insuring that the message reached at the destination of the channel, the ID of the channel is removed from all records of temporary store. Since storing messages in temporary store is not realistic, our algorithm deletes records containing one channel ID when that channel ID is removed from the record.

5.3.4. Dynamic Graph

Although the solution proposed so far provides guaranteed delivery of message in the presence of mobility, the necessity of knowing the nodes' neighbor in advance is sometimes unreasonable in dynamic environment of mobile software agents. Furthermore, dealing with all nodes including nodes which may not be visited by mobile software agent, will make the algorithm somewhat complex. Particularly, in the existence of disconnection, the time to update nodes which may not be visited by the agent, will have an effect on the efficiency of the algorithm.

As in the case of the algorithm for static Network graph, our algorithm for dynamic network graph adapts Murphy's algorithm. However, the adaptation is not only on handling disconnection but also on the construction of the network and the way messages are delivered and sent. These modifications is needed to remove the following drawbacks of Murphy's algorithm

- Murphy's algorithm for dynamic network graph can't be used to guarantee delivery of message in the existence of disconnection as discussed on previous section, since it is extended from her algorithm for static network

- The algorithm sometimes inhibits mobile agent from movement and delay the activation of the channel to solve two difficulties (section 5.2.) called destination ahead of source and source ahead of destination respectively.

Solutions for the problems

Removing the two problems discussed above will make the algorithm to work without inhibiting agents from movement and without delaying activation of a channel. As discussed on section 5.2 the two difficulties result from the fact that the network is constructed by considering all agents of a single source. Therefore, to remove the drawbacks, network graph should not be constructed by considering all agents. Considering different network graphs for each agent will solve the problem.

The algorithm that we develop considers different network graphs for each agent. As Murphy's algorithm, our algorithm assumes messages as well as the agent are from the same source. However running the algorithm concurrently will provide guaranteed message delivery in the existence of fault.

In the next section, we present first the algorithm by assuming the existence of only one agent exists in the network and the extended algorithm that doesn't make such assumption as solutions for the problem above.

Basic Algorithm For Single agent

The algorithm presented in this section, works in the same way as Murphy's algorithm except that it assumes only one agent in the network and handles disconnection. Here the two difficulties, source ahead of destination and destination ahead of source cannot occur. This is because there is only one agent in the network and the graph is only constructed by it.

There are problems resulting from the fact that the network is constructed dynamically. One of the difficulties is, the node doesn't know all the incoming as well as the outgoing channels before the agent is destroyed. For example, in figure 5.10, at time t1, node 1 had three incoming channels and at t2, it has three incoming channels. The other difficulty is all the channels couldn't be constructed at the same time; so it is not possible for some messages to be transported by all channels. For example, in figure 5.10, the channel that connects node 2 and node 1 cannot transport m1. The third difficulty is it is very difficult to delete a message from the node. For example in figure 5.10, if the message at node 2 were cleared at t1, agent 1 can't get m1 at t2. This problem mainly affects the clearing of messages at a node since clearing of message depends on the state of the incoming channels. Therefore, modifying the way messaged cleared at the node will removes the problem resulted from the difficulties discussed above.

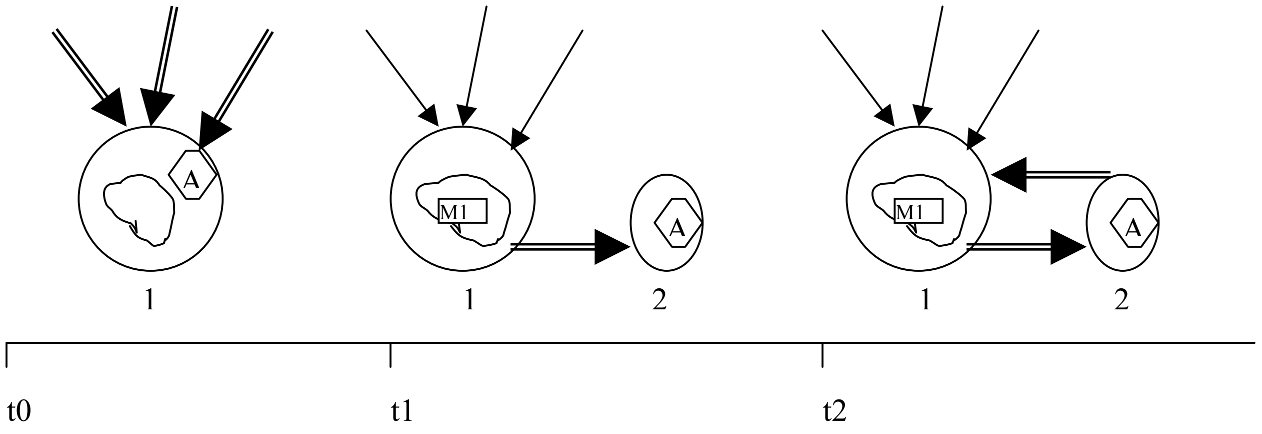


Figure 5-10 The effect of constructing the network graph dynamically

To remove these problems we make

- the network graph to start from the home of the agent which can be extracted from the agent name,
- the node to hold all messages arrived at a node,
- messages to have a sequential ID which indicate their order and
- the agent to retrieve the messages by themselves according to the message sequential ID.

In our algorithm for dynamic network when the message arrives at a node while the node holds another message, the message will also be considered as active message in the node rather than being added in the buffer queue of the channel.

Message delivery in this algorithm is done as follow; When a message arrives at a node, the node delivers it to the agent, to which the network graph is constructed, and the message broadcast in all outgoing channels of the network graph. When an agent arrives at the node, it is allowed to retrieve the messages from the node.

Since message is broadcasted with all outgoing channel that has been already constructed, some node may not contain all messages (figure 5.11). In this algorithm, since the agent constructs the channels, this will not create problem. This is because the agent will get the messages broadcasted before constructing the new channel.

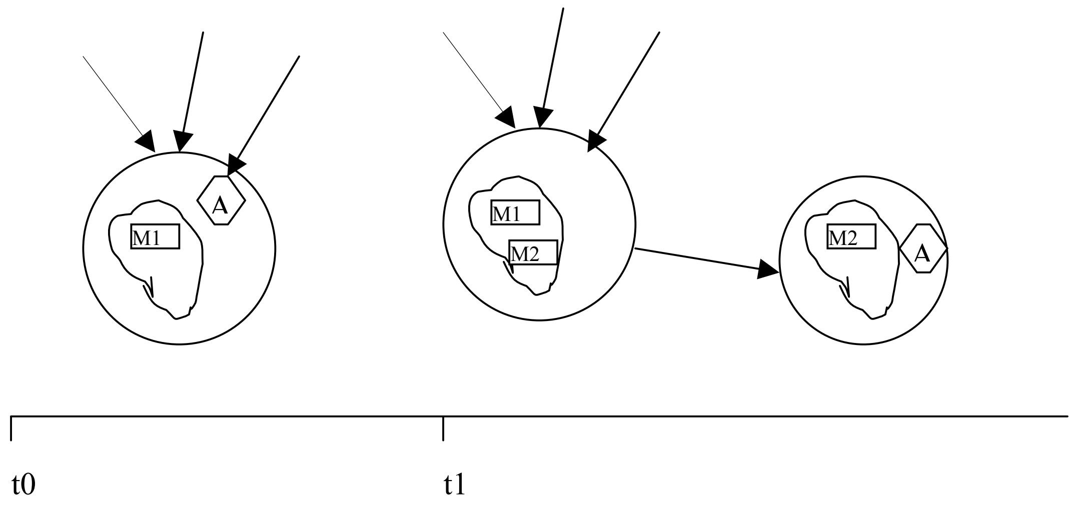


Figure 5-11 Message content at different node is different

Clearing the message at the node is important because storing the message indefinitely is not a realistic solution. In addition, storing the message after the agent is destroyed is not necessary. Therefore, clearing of messages on the node should be done after the agent is destroyed. The network graph is no more important after the agent to which it is constructed is destroyed. Therefore the algorithm should remove the network graph, when the agent to which the graph is constructed, is destroyed. Clearing of the message can be done when the network graph is removed.

Messages stored at a node will be removed when the network graph is removed. The network graph is removed when the agent for which the network graph is constructed, is destroyed. When a node destroys the agent, it informs the node that creates the agent. The node, which creates the agent, deletes the messages at the node, removes itself and sends the information to its successors and the successors do the same as their predecessor.

As in the case of the static network graph, in dynamic network graph disconnection may occur at any time. Therefore handling disconnection is important. Our algorithm for dynamic network handles disconnection in the same way as in the case of static network graph, on which multiple messages are assumed in the network graph. This means, by changing the way of message broadcasting as discussed in section 5.4.

Extended Algorithm for Multiple agents

The algorithm for single agent can be run concurrently to guarantee delivery of message to multiple agents. Concurrency will be achieved by considering different network graph for each agent. Here, one node can be a part of more than one network graph. Since message source node can be a part of two network graphs, it may be difficult to the node to know which network graph to use to broadcast the message. Since our algorithm deals with delivery of message to a single agent, it is possible to know to which agent the message is sent. Therefore, in our algorithm message source node will use the network graph constructed by an agent to which the message is sent.

5.5. Summary

In this chapter, we presented an algorithm that provides a guaranteed delivery of message in the existence of disconnection. The algorithm adapts Murphy's algorithm and works in the existence of disconnection.

The algorithm that we have designed, assumes the channels are FIFO. Even if in reality, physical channels are not usually FIFO channels, there are protocols like TCP, which can be used to provide virtually FIFO channels. Therefore, it is possible to implement the algorithm by using the existing technology.

6. Implementation

In this chapter, the prototype and the components needed to implement the prototype will be presented. The prototype has two major components (subsystems). One of the subsystems is to provide reliable message delivery to mobile software agent by implementing the algorithm presented in chapter 5, and the other subsystem simulates a mobile software agent system. Simulation of mobile software agent is needed because developing the mobile software agent system is beyond the scope of the study.

Experimental settings to test the prototype and analysis of the experiment will be presented in this chapter. The prototype is implemented on wired LAN environment. Since disconnection is rare in such environment, we decided to physically simulate disconnection, by unplugging cables.

In section 6.1, we will discuss about the subsystem that implements the algorithm presented in chapter 5. In section 6.2, we discuss about the mobile software agent simulator. In sections 6.3, 6.4 and 6.5, the implementation details of the prototype, the experimental settings and analysis of the result will be presented. Finally, we will discuss the feasibility of using our system in real mobile agent systems.

6.1 Communication System

The algorithm discussed in the previous chapter can be applied to develop a subsystem that can provide a guaranteed message delivery to mobile software agents. The objectives and the requirements of the subsystem that implements the algorithm presented in chapter 5 are discussed below. From now wards, we will refer the subsystem as **CommSystem.**

Objectives

The objectives of developing a communication system for mobile software agents are:

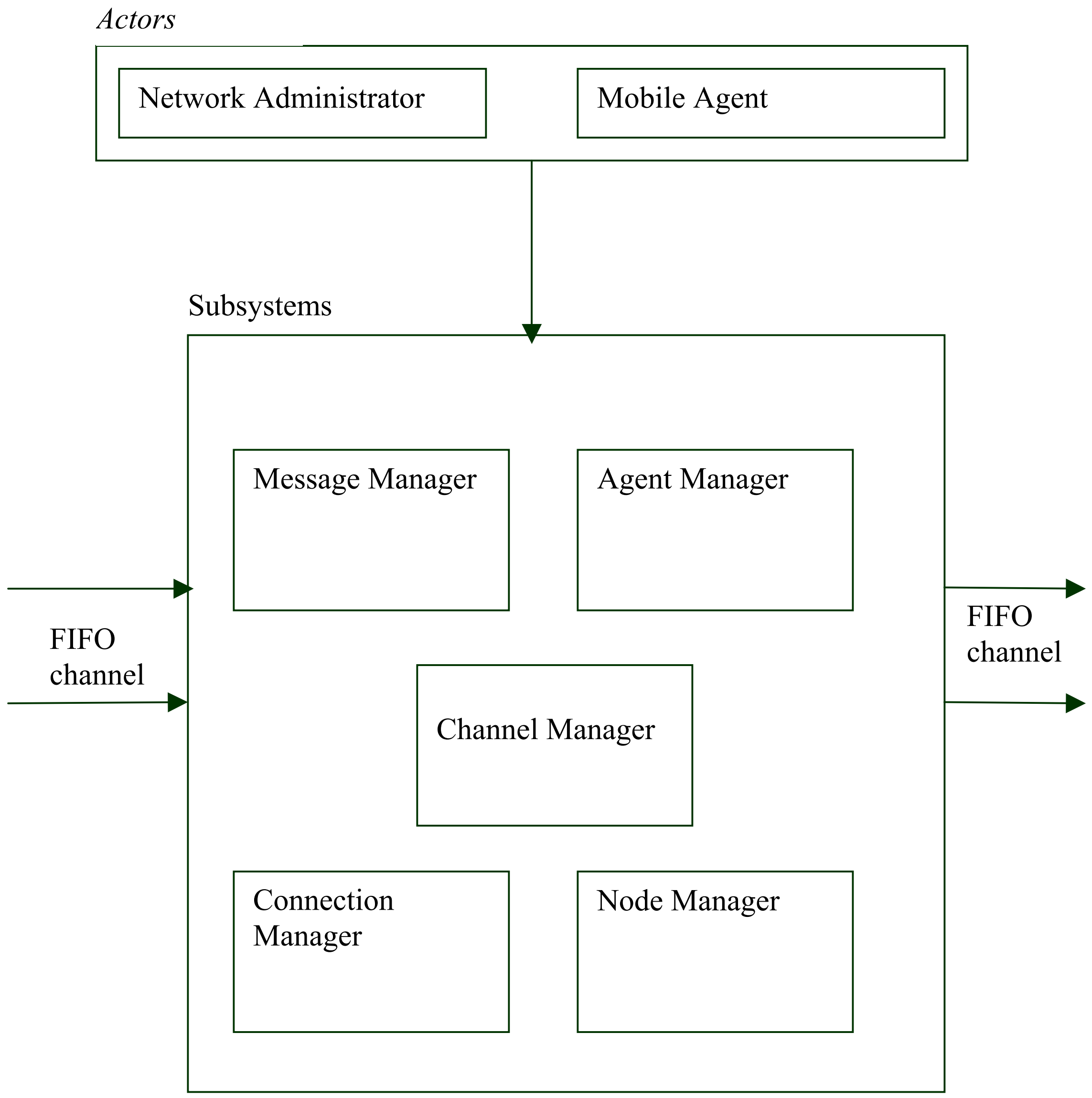
- To enable agents to communicate transparently
- To provide a guaranteed delivery of message to mobile software agents by applying the algorithm presented in this paper.

Requirements

The system should have the following properties

- It should be flexible to support dynamic or static network graph
- It should be able to manage channels according to the type of the graph.
- It should be able to manage the messages in the node as discussed in the study
- It should be able to deliver messages to mobile software agent.
- It should be able to broadcast the message as discussed in the study

Data Flow Diagram (DFD) of CommSystem



Description of the DFD of CommSystem

Actors

Mobile Software Agent: agents, which use the system to send and receive messages.

Network Administrator: a person who set network type and channels

Physical Component

FIFO-Channel: it is a channel used to send and receive messages from the neighboring nodes. The FIFO channel can be implemented by using TCP socket.

Subsystems

Message Manager is used to manage the active message. In other words, it is used to keep copies of messages in the node and to remove the messages from the node as discussed in the study. This subsystem will also enable nodes to broadcast messages accordingly.

Connection Manager is used to detect disconnection and reconnection of channels, and it makes other subsystems to know the disconnection and reconnection of a channel.

Agent Manager is used to manage agents. It delivers messages to the agent by using different mechanisms provided by the mobile software agent system.

Channel manager is used to manage channels. It keeps track of the state of the channels. The channels can be used to transport messages and agents. The channel manager passes the agent transported to agent manager and the message transported to message manager.

Node Manager is used to setup channel and create active message storage, temporary storage and buffer queues.

6.2 Mobile Agent System Simulator

As discussed in the beginning of the chapter, we need to develop the system that simulates mobile software agent system. To simulate the mobile software agent system, we tried to identify the important characteristics of the mobile software agent system for the study. Accordingly, the following key characteristics of the mobile software agent that are needed for the study are identified

- The mobile software agent should give names to agents such that each agent has unique name and the name should contain the name of the node that has created the agent
- The node should identify the agent hosted there
- The agent should to be able to move freely
- The agents should be able to communicate to each other

We have simulated the agent as a message. It is represented by its name which is a text. The message sent by the agent will have a name (ID), content, the name of the sender agent and the name of the receiver agent. The creator node gives a name to the agent and to achieve uniqueness throughout the network, the agent name will contain the node name.

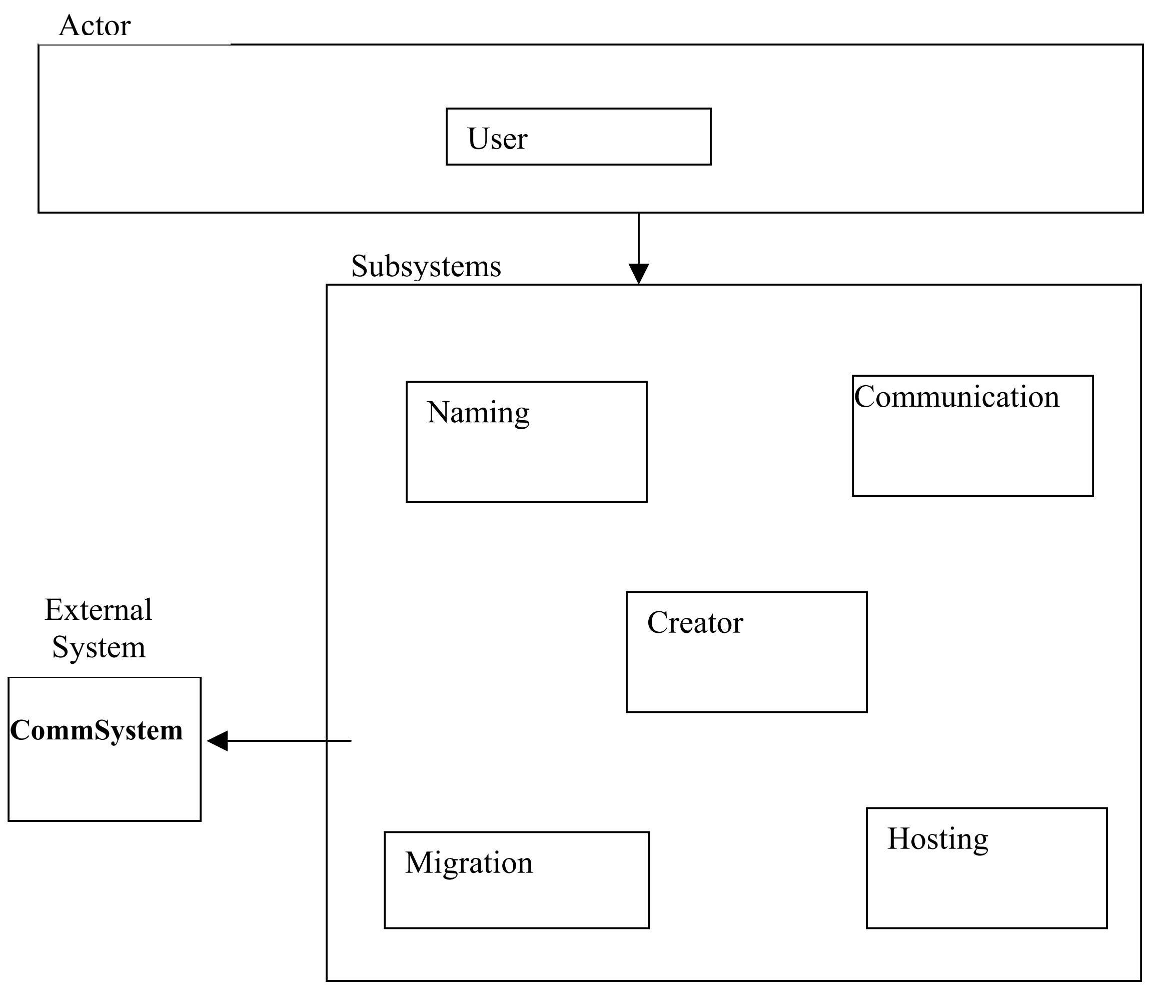
Each node can register, deregister and search mobile software agent hosted. When an agent arrives at a node, it will be registered at a node and when it migrates from the node, it should be deregistered from the node. In addition, the node should be able to search an agent by accepting the agent name.

The movement and the communication of the agent is simulated as follow”

- movement :-a hosting node decides randomly which agent should move to which node and sends the selected agent to the selected node.
- communication:- a node is designed to provide an interface which lists the agent previously and currently hosted and the user composes the message and decides to which agent the message should be sent to.
- message delivery: the system will display the message and the agent together when a message is arrived.

Generally, the mobile agent simulator which is needed to demonstrate the developed algorithm on the previous chapter, should simulate the agent, the free movement of the agent and communication among the agents.

The Data Flow Diagram for the simulated software agent system



Description of the DFD of Mobile Software Agent System Simulator

In the DFD the actor, the external system and, the subsystems of the system, which simulate the mobile software agent, are shown. The user is an actor, which involves on creating an agent and sending a message to an agent. The **CommSystem** is an external system, which is presented on the above chapter. The subsystems shown in the DFD as part of the system are described as follow.

- Creator subsystem: is used to create and destroy agents according to user commands,
- Naming Subsystem: is used to give unique name to the agent
- Migration subsystem: is used to decide when and where the agent leaves the node.
- Hosting subsystem: is used to register agents when hosted and to deregister them when they leave. This subsystem will also provide searching service.
- Communication subsystem is used to enable users to send messages to an agent.

6.3 The prototype

According the requirements discussed in the above section, a prototype for static network graph is developed. The prototype implements the **CommSystem** and the system which simulate the mobile software agent system. It is developed to provide delivery of messages of a single source.

The prototype has two versions. The first version doesn't assume disconnection and have component to test disconnection. The second version works in the existence of disconnection and have a component that detects disconnection and reconnection of a channel.

As discussed in the previous section, channels are implemented by TCP socket. Using socket as a channel introduce a difficulty on detecting disconnection. It is not possible to get the connection status of the channel from TCP, since TCP does not provide any notification when a network link is lost. Therefore the successor made to send a special message (we call it heartbeat), to the predecessor periodically. Period will be set by the network administrator. The disconnection manager subsystem will detect the disconnection of outgoing channel by observing the heartbeats.

The User Interface for the prototype

The prototype provides user interface which includes interface to create agents, send messages to an agent and display the messages that are sent to hosted agent. The interfaces used are described as follow.

1. Interface used to create agents and send messages to an agent

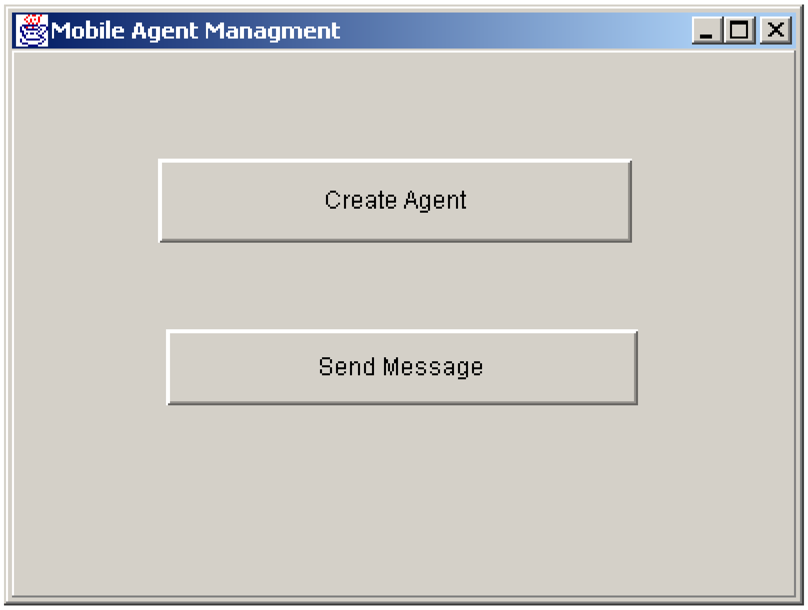


Figure 6-1 User interface used to create and send message

In figure 6.1 the interface that can be used by the user to create an agent and send a message to an agent in the network without specifying the location of the agent is displayed. When the user clicks the create button the agent will be created and it becomes active and when the user click the send message, the dialog box shown in figure 6.2 will appear and the user can use it to send messages to an agent.

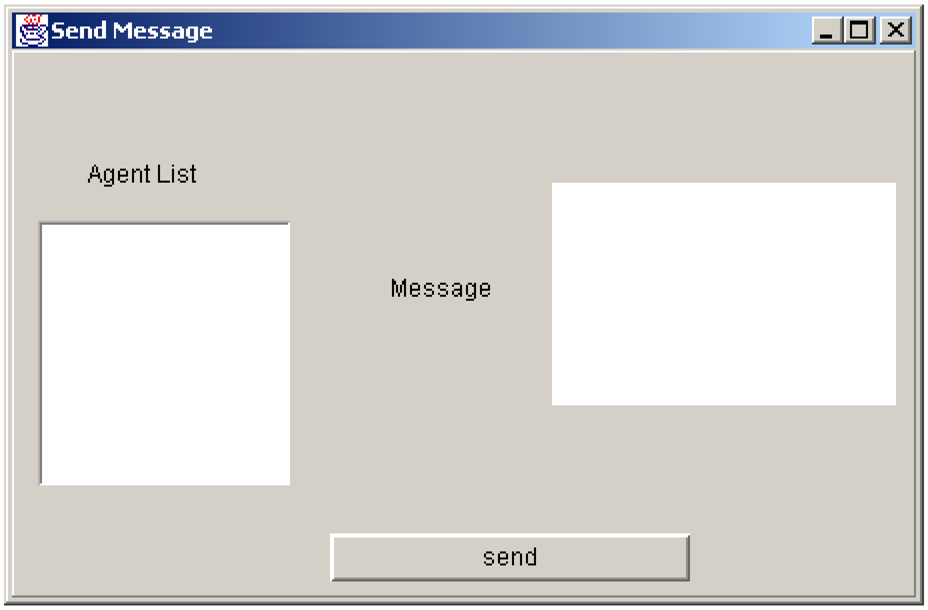


Figure 6-2Communication interface

2. Interface used to post the message send to an agent

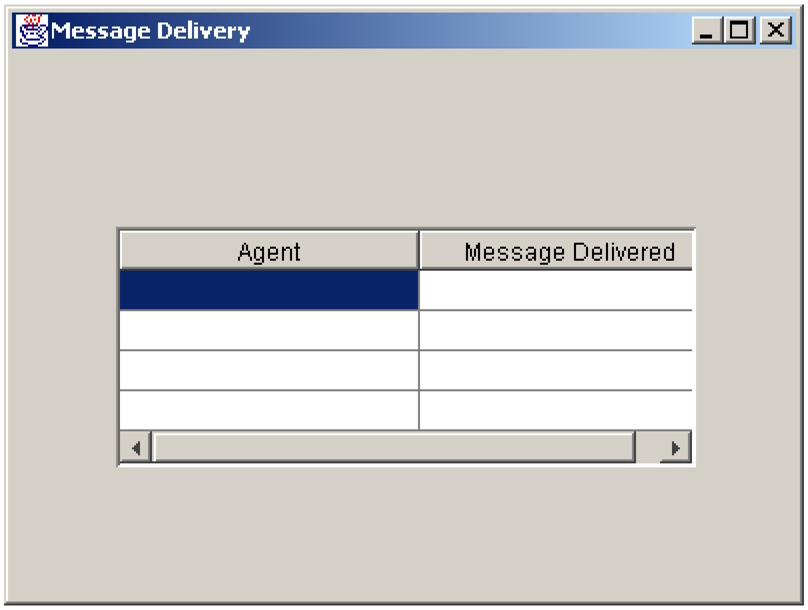


Figure 6-3 Interface used to post message send to an agent

When a message arrives at a node and if it is sent to an agent, the hosting node will post agent name and message content by using the dialog box displayed in figure 6.3

6.4 Languages, Tools and Experimental settings

In the previous sections, we have described the prototype used to demonstrate the study. In this section, we will describe the programming language and the tools in the development of the prototype, and the excremental settings to test the prototype.

The choice of the programming language is one of the important issues in system development. To that end, we have selected java to develop the prototype. This is because of the experience of the researcher, free availability of java and easiness of network programming in java.

The other important issue, that determines the efficiency of the system development is the selection of development tools. Some of the tools that we use to develop the prototype are j2sdk 4.2 and java IDS: Jbuilder and Jbeans.

We have used three computers in the local area network of the department of computer science, AAU. These computers can have any capacity. Since the prototype is demonstrated in wired local area network and disconnection is rare there, we have decided to simulate disconnection physically, by unplugging cables.

6.5 Analysis of the Result

By using the prototype, we were able to demonstrate that the developed algorithm for static network graph works. The demonstration is done into three steps. In the first step, the prototype that implements Murphy's algorithm is tested, in the second step, the same prototype is tested in the presence of disconnection and in the third step, the prototype that is modified to have connection manager, is tested in the presence of disconnection.

In both first and third steps the algorithm successfully. In the second step, when a cable is unplugged and then plugged, the prototype stops working.

A guaranteed message delivery by the prototype that implements Murphy's algorithm, is fully achieved in the absence of disconnection (i.e., all messages that are sent are broadcasted in every node in the network, and the messages are delivered to all agents).

The test results of the prototype that implements our algorithm (have a disconnection manager to tolerate disconnection) are the following. All messages that are sent, was successfully broadcasted in every node in the network graph. However, some agents didn't get all the messages broadcasted, because they are damaged by the disconnections. We can conclude that an agent is damaged if it is not displayed in any node. This is because the prototype will display the agent in the node. The failure of message delivery due to damage of the agent cannot be counted as a failure of the algorithm, since in the algorithm; it is assumed that the channels should not fail while transporting the agent.

6.6 Discussion

Form the demonstration of the developed algorithm, we have arrived at the following remarks. It is very difficult to implement the communication system without implementing the mobile software system since the algorithm requires that the agent and the message should be transported together. Therefore, implementing our algorithm requires the development of a mobile software agent system, which is beyond the scope of this study. However, we are able to demonstrate the algorithm works by using a system that simulate mobile software agent.

To implement the algorithm without simulation, what we need is the mobile software agent that uses FIFO channel to transport message and agent. Since most of java mobile agents like Aglet develop agent migration on top of socket, using FIFO channel to transport the message is not difficult. In addition the mobile software agent should provide a naming service, which gives unique name to each agent. Since this is also already provided most of mobile agent, this is also not difficult. Lastly to implement the algorithm, the mobile software agent system needs to have a mechanism to deliver message to the agent. This also can be achieved by using Java space.

7. Conclusions

7.1 Conclusions

Mobile software agents are becoming an important tool to develop distributed applications particularly in environments with unreliable connections is unreliable and low bandwidth. Mobile Software agent technology is emerging as an important area for computing research. It is posing many challenges which should be overcome in order to use this technology to develop distributed applications. Guaranteed message delivery in the existence of fault is one of the challenges that need to be solved.

In this thesis we have developed an algorithm that guarantees the delivery of message to highly mobile agents in the existence of fault, more specifically disconnection. We also presented the implementation issues of the algorithm and demonstrated the algorithm for the static network by developing a prototype. Since in our algorithm we required to transport the agent and the message together in a FIFO channel, we were unable to use the existing mobile software agent system to develop the prototype. However, we developed a system that simulate mobile software agent to demonstrate the algorithm that has been developed.

7.2 Contribution of the study

The contributions of the study are: (1) Murphy's algorithm is extended to work in the existence of disconnection. (2) The newly developed algorithm removes Murphy's algorithm drawbacks (Murphy's algorithm drawbacks are inhibition of the movement of agents and delays of activation of channels.) (3) The implementation issues of the developed algorithm are identified.

7.3 Future Work

The algorithm is tested on the available wired local area network. Thus, further validation should be made on the effectiveness of the algorithm on different LANs and WANs. In addition, since only the version the algorithm for static network graph has been demonstrated, the version of algorithm for dynamic network should be demonstrated. Since the algorithm correctness is not proved in this thesis, it should be formally proved.

References

[1] K. Mani Chandy , Leslie Lamport, Distributed snapshots: determining global states of distributed systems, ACM Transactions on Computer Systems (TOCS), v.3 n.1, p.63-75, Feb. 1985

[2] Mikael Berg “Software agent frame work technology” Linkoping University 2000

[3] Roberto Silveira Silva Filho “The Mobile Agents Paradigm” Department of Information and Computer Science University of California, Irvine 1998

[4] T. Middelkoop and A. Deshmukh "Mobile Agents for Collaborative Design and Analysis," in 1997 International CIRP Design Seminar,Oct. 1997

[5] L. Levison, W. Thies, and S. Amarasinghe, “Providing web search capability for low-connectivity communities,” Proceedings of the 2002 International Symposium on Technology and Society, Raleigh, North Carolina, 2002

[6] Freeman Yufei Huang “Communication Infrastructures and Protocols for Mobile Agent” December 2000, accessed on October 12 2003

[7] Anand R. Tripathi Neeran M. Karnik Manish K. Vora Tanvir Ahmed Ram D. Singh “Mobile Agent Programming in Ajanta” Proceedings of the 19th International Conference on Distributed Computing Systems 1999

[8] Jiannong Cao, Xinyu Feng, , Jian Lu, and Sajal K. Das "Design of Adaptive and Reliable Mobile Agent Communication Protocols" Proceedings of the 22nd International Conference on Distributed Computing Systems 2002

[9] DavidWong, Noemi Paciorek, and Dana Moore, "Java-based mobile agent", accessed on October 12,2003

[10] Stefan Fünfrocken "Transparent Migration of Java-Based Mobile Agents" Proceedings of the Second International Workshop on Mobile Agents 1998

[11] Stefan Pleisch, Andre Schiper "Modeling Fault-Tolerant Mobile Agent Execution as a Sequence of Agreement Problems" Proceedings of 19th IEEE Symposium on Reliable Distributed Systems 2000

[12] Giacomo Cabri, Letizia Leonardi, Franco Zambonelli "Engineering Mobile-agent Applications via Context-dependent Coordination" IEEE Transactions on Software Engineering, 2002

[13] Amy L. Murphy "Enabling the rapid development of dependable of development applications in the mobile environment" Washington University August, 2000

[14] Joachim Baumann, Fritz Hohl, Nikolaos Radouniklis, Kurt Rothermel, and Markus Stra\sser "Communication concepts for mobile agent systems" In Proceedings of the First International Workshop on Mobile Agents (MA '97), pages 123-135, Berlin, Germany, April 1997. Springer Verlag.

[15] Pawel T. Wojciechowski, Peter Sewell “Nomadic Pict: Language and Infrastructure Design for Mobile Agents” Computer Laboratory, University of Combridge” 1999

[16] Flávio Morais de Assis Silva, Raimundo José de Araújo Macêdo “Reliability Requirements in Mobile Agent Systems” Proceedings of the Second Workshop on Tests and Fault Tolerance (II WTF 2000), 15th-16th july 2000

[17] Peter Sewell, Pawel T. Wojciechowski, and Benjamin C. Pierce. “Location independent communication for mobile agents: A two-level architecture” In Internet Programming Languages, LNCS 1686, pages 1--31. Springer, 1999.

[18] Robert S. Gray “Agent Tcl: A flexible and secure mobile agent system”, 30 June 1997

[19] Yolande Berbers, Bart De Decker, Wouter Joosen “Infrastructure for mobile agents” Department of Computer Science, KULeuven 1996

[20] V. Roth. “Scalable and secure global name services for mobile agents” 6th ECOOP Workshop on Mobile Object Systems: Operating System Support, Security and Programming Languages June 2000

[21] Jonathan Dale, David C. DeRoure “A Mobile Agent Architecture for Distributed Information Management” Proceedings of the International Workshop on the Virtual Multicomputer 1997

[22] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, and B. Peet. “Concordia: An infrastructure for collaborating mobile agents. In Proceedings of the First International Workshop on Mobile Agents (MA '97), volume 1219 of Lecture Notes in Computer Science, Berlin, April 1997

[23] Asis Unyapoth, Peter Sewell “Nomadic Pict: Correct Communication Infrastructure for Mobile Computation” ACM SIGPLAN Notices 2001

[24] Sebastian Fischmeister, Giovanni Vigna, Richard A. Kemmerer “Evaluating the Security of Three Java-Based Mobile Agent Systems” Proceedings of the 5th International Conference on Mobile Agents 2001

[25] Frederick Knabe, “An Overview of Mobile Agent Programming”, Selected papers from the 5th LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages, p.100-115, June 24-26, 1996

[26] Antonio Carzaniga, Gian Pietro Picco, Giovanni Vigna “Designing distributed applications with mobile code paradigms” Proceedings of the 19th international conference on Software engineering 1997

[27] E.W. Dijkstra and C. Scholten. “Termination detection for diffusing computations” Information Processing Letters, 1980.

[28] L. Eggert “TCP Abort Timeout Option” The Internet Society (2004), 2004, accessed on May 27,2004

[29] Robert Gray, David Kotz, Savrab Nog, Daniela Rus, Georgo Cybenko "Mobile Agents: The Next Generation in Distributed Computing" Darthmouth College 1997

[30] Christos Bohoris "Network performance management using Mobile Agents" Univeristy of Surrey, June 2003

[31] David Kotz, Robert S. Gray "Mobile Agents and the Future of the Internet" In ACM Operating Systems Review 33(3), August 1999, pp. 7-13.

[32] Giacomo Cabri, Letizia Leonardi, Franco Zambonelli, "Context-dependency in Internet-agent Coordination", Università di Modena e Reggio Emilia, 2000

[33] Clements, P. E., Papaiannou, T. and Edwards, J. M., "AGLETS: Enabling the Virtual Enterprise", Proc. the 1st International Conference on Managing Enterpriser - Stakeholders, Engineering, Logistics and Achievement (MESELA '97) , July 1997

[34] Giacomo Cabri, Letizia Leonardi, Franco Zambonelli "Mobile-Agent Coordination Models for Internet Applications" University of Modena and Reggio Emilia, 2000

[35] Carriero & D. Gelernter, "Linda in Context", Communications of the ACM, 1989

[36] J. Bayman, M. Hohle, K. Rothernel, M. Straber "Mole – Concepts of a mobile agent system", Kluwer Academic Publishers Hingham, MA, USA , 1998