

Honours Project 2003

Mobile Agents in Ad-hoc Networks

Submitted in partial fulfilment of the requirements of Napier
University for the degree of Bachelor of Science with Honours
in Computing

Supervised by Dr. Bill Buchanan
School of Computing

April 2003

Authorship declaration

I, Alexandre MANGUER, confirm that this dissertation and the work presented in it are my own achievement.

1. Where I have consulted the published work of others this is always clearly attributed.
2. Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work.
3. I have acknowledged all main sources of help.
4. If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself.
5. I have read and understand the penalties associated with plagiarism.

Acknowledgment

I would like to thank my supervisor, Dr Bill Buchanan, for all the help he gave me throughout the project, and also for providing the subject of this project which proved to be very interesting to study, and very valuable for my experience. I also would like to thank all the people that advised me in any part of my project.

Table of Contents

Authorship declaration	2
1 Introduction	8
1.1 Aims and objectives	8
1.2 Generalities on distributed computing	8
1.3 Ubiquitous and nomadic computing	10
1.3.1 Ubiquitous computing	10
1.3.2 Context-aware computing	10
1.3.3 Mobile Agents Paradigm	10
2 Overview of mobile agents	12
2.1 Introduction	12
2.2 Agent-based systems	12
2.3 Main advantages	12
2.4 Shortcomings and limits of agents	13
2.4.1 Agents and security issues	14
2.5 Wireless LANs	14
2.5.1 Different Wireless technologies	14
2.5.2 IEEE 802.11	14
2.6 What mobile agents can offer to wireless	15
3 Programming Mobile Agents	16
3.1 Introduction	16
3.2 Generalities on network programming	16
3.2.1 Client-Server paradigm	16
3.3 Java network programming models	17
3.3.1 CORBA	17
3.3.2 RMI	18
3.4 Agent Platform evaluation and comparison	19
3.4.1 IBM Aglets	20
3.4.2 Grasshopper	20
3.4.3 Tracy	20
3.5 Agent-based system architecture	21
3.6 Selected environment	21

3.7	Grasshopper environment	22
3.7.1	Introduction	22
3.7.2	Grasshopper's Requirements	22
3.7.3	The architecture of Grasshopper	22
3.7.4	The General User Interface (GUI)	22
4	Implementation involving Mobile Agents and Wireless	24
4.1	Aims	24
4.2	Issues involved in the routing on wireless networks	24
4.3	Study of the research undertaken in the domain	24
4.4	Design of a model of mobile routing	25
4.5	Mobile agent migration	28
4.6	The process of coding	28
4.6.1	Grasshopper static agent programming	28
4.6.2	Read a database: JDBC with Access	29
4.6.3	Grasshopper mobile agent programming	30
4.6.4	Pop up to get IP Addresses	31
4.6.5	Moving the agent	31
4.6.6	Contact local agency and finding static agents	31
4.6.7	Create and establish proxies of the server agent	32
4.6.8	Final actions	33
4.7	Elements of implementation	33
4.7.1	The Agency Explorer window	33
4.7.2	The console window	33
4.8	Conclusion	33
5	Tests	36
5.1	Introduction	36
5.2	Configuration for the tests	36
5.3	Tests results	36
5.3.1	Series 1: Migration timing for a four host system with a small database	37
5.3.2	Series 2: Migration timing for a four host system with a large database	38
5.3.3	Series 3: Migration timing for a four host system with no database	40
5.4	Analysis of the results	41

6 Conclusion	42
6.1 General	42
6.2 Results	42
6.3 Future application of mobile agents	43
6.4 Further work	43
7 Appendices	44
7.1 Install Grasshopper	44
7.2 Grasshopper settings	44
7.3 JDBC Setup on a machine	44
7.4 Jcreator – Java	44
7.5 Static Agent Code	46
7.6 Mobile Agent Code	47
8 References	51
9 Gantt	54

Abstract

The aim of project has been the evaluation of mobile agents for their usage in the ad-hoc network, especially related to wireless applications. It integrates with research being conducted in the School of Computing at Napier, on the usage of agent-based systems to provide on-demand routing through ad-hoc wireless networks. This reports presents a study of mobile agent programming over ad-hoc networks, and especially wireless ones.

Mobile Agents and wireless networks are two cutting-edge technologies that will provide enhancements for increased connectivity and communicability. Mobile agents are application programs that not only have the ability to move by themselves from machine to machine autonomously, but also the capability to interact with their environment. Wireless network technologies use radio communication networks that have for main goal the providing of computers, handheld devices or even mobile phones with rapid connectivity. In a wireless network, any device can easily join or quit a work group without the need of a physical connection.

Obviously, these two technologies are very fascinating but their interest is increased when they are combined. Indeed, mobile agents, by providing a new paradigm of computer interactions, give new options for developers to design applications-based on computer connectivity. Because of this, they seem particularly well fitted to move around wireless networks in a more elegant manner than common applications that were based on static models, such as client-server architecture.

In this report, mobile agents were used to build a prototype of routing system specific to wireless networking needs. The aim was to be able to dynamically set a path whenever a computer wants to access resources remotely located over the wireless network, and so benefit from a greater mobility to analyse network status directly on the host, rather than remotely over the network. More mobility must not hide the fact that bandwidth is an important factor for networking, thus it is a major focus of for this project.

In general, the project evaluates the main mobile agent development systems, such as IBM Aglets, Tracy and Grasshopper, of which Grasshopper was chosen as it supports mobile devices, such as PDA's, and it has extensive documentation. The model uses static agents which interface to local databases, and mobile agents which migrate around the network and communicate with static agents. This method enhances the security of the overall system, as mobile agents are not allowed to interface directly with the hosts.

Mobile and static agents have been developed using Java JDK 1.4, and tests show migration timings for differing sizes of database sizes, and different migration strategies. Each host uses a JDBC database to store data, and uses a proxy for asynchronous communication between the static and the mobile agent. The conclusions outlines the general benefits of mobile agents, and recommends future work.

Keywords: Client-server, distributed computing, mobile agents, static agents, agent migration, Grasshopper, Tracy, IBM Aglets, Java JDK 1.4, wireless networks, ad-hoc routing, JDBC database, message passing, asynchronous communications, proxy.

1 Introduction

1.1 Aims and objectives

The aim of the project is to investigate the usage of mobile agents over a wireless, ad-hoc network, especially in creating a system which allows mobile agent to be dispatched around a network, and communicate with local static agents. The main stages of the project have been:

- **Research:** Investigation of mobile agent resources and development systems, along with the investigation of their migration of wireless networks.
- **Design:** Design of a models for mobile agent migration over wireless networks.
- **Implementation:** Implementation of a mobile agent which migrates over a wireless network, and communicates with a static agent.
- **Analysis:** Results of migration strategies. Analytical tests of mobile agent performance over wireless networks.

This work integrates with work being conducted in the Distributed Systems and Mobile Agents (DSMA) research group at Napier. The DSMA group are researching into methods of providing dynamic routing over wireless network [26, 31, 32]. One of the method used is to use static agents to determine metrics that could be used for routing data through and ad-hoc wireless network, and use mobile agents to collect this information, and return it to a routing agent. This application is known as ad-hoc, on-demand distance vector (AODV) routing [27]. Migas [29] outlines that the aim of the research is to:

... design and implement a novel routing scheme based on mobile agent technology in wireless ad-hoc networks that will provide the following benefits. Maximize network **performance**, **scalability**, **provide end-to-end reliable communications** and **reduce possible delays**, and **minimize losses** that may incur due to bit-errors and handoffs.

1.2 Generalities on distributed computing

Client-Server architecture is widely spread over the Internet, and has been successful in most Internet applications. Nevertheless, one of the major issues at this time is that servers tend to become sources of bottlenecks. Indeed, in connection-oriented client-server systems, most frequent on the Internet, clients need to be in a blocked state in order to wait for server replies. When a saturated server has to deal with too many requests simultaneously, it will normally queue clients until it has processed all previous requests.

In the client-server model, a server waits for a connection with a client, which will then request services from it, such as file services, WWW services, remote login services, and so on. The client-server model has the advantage that it is easy to implement, but it has many weaknesses. Until recently it had still it had no real competitors, but this has changed with the emergence of mobile agents, which is a new paradigm that presents a better way of sharing resources at a global level.

It is normally up to servers to execute Internet applications; this is true when we talk about Common Gateway Interface (CGI) languages, Java Servlets (including Java Server Pages, standing for JSP), Professional Home Page (PHP), and so on. All these applications are qualified of server-sided. However, over the recent years, the trend has evolved differently and new technologies appeared following on from concepts such as Java applets. Indeed, Java applets offered a first step ahead in the area. They are little applications that are downloaded from a server, but then executed locally, thus reducing the amount of processing power needed on the server. However, this still not resolved the problem of bottlenecks. With mobile agents, the concept is different. Common applications can use different forms of inter-process communication:

- **Sockets.** This is where two application program use a common TCP port, on which they communicate using uniquely defined sockets. The combination of the TCP port for the source and the destination defines a unique socket. For example a host may communicate locally with a port of 1001, with an FTP server which is using port 21. The socket is thus defined by the combination of 1001 on one host, and port 21 on the server.
- **Shared memory.** Two program can communicate using a shared area of memory. This is a fast memory of communicating, but is likely to be specific to a certain type of operating system, or hardware architecture.
- **Files.** This is a relatively slow method of passing information between applications, where one process creates a file which is then read by another process.
- **Message passing.** This method uses messages passed over the operating system. Microsoft Windows uses a message passing system for inter-process communications.
- **Remote procedure call (RPC).**
- **Remote method invocation (RMI).**
- **CORBA.**

To communicate, agents mainly use message passing, RPC, and RMI (for Java systems only). With these, objects are able to access remote methods in the same way as they would locally. Intelligent agents are very useful assistants that can work for users, without them actually operating on them consciously [23–25]. Basically, the idea behind all the technical concepts is that mobile agents move and get resources for users, even before they ask for them. For both developers and the business market and developers, this means that server's loading will be reduced as agent can move from client-to-client. The server simply remains a central point of access for important data, a set of file servers for instance. The process itself is transferred onto all clients, all workstations needing to access the information.

1.3 Ubiquitous and nomadic computing

1.3.1 Ubiquitous computing

Ubiquitous and context-aware computing are two concepts bound together and are likely to lead to the next way of computing. Rather than a technology itself, ubiquitous is the name given to an innovative paradigm in which computers are everywhere and nowhere in the same time. Thus, the current location of the information/applications/resources loses its relevance.

With the revolution in networking primarily, and also with the next generation of mobile devices, ubiquitous computing aims to unify the different sources of information wherever they are through a single flow, transparently to the user. An example of this relates to the Internet itself, where computers around the world from different types provide with their own sources of information; the end user gets this through a sole interface, the WWW, and has the ability to access this incredibly huge amount of data. Other examples of this concept are of course numerous, they all tend to achieve one common goal: forget computers and just people live. This attractive slogan that could remind Bill Gates' own theories of the information highways is well accepted, everybody tends to agree on the fact that now that computers have gained their place in our society they should rather merge into our lives so that they do not look like breaking into them. Mobile devices progression shows that this evolution will be easier with the benefits of smaller devices, so tiny that they can always stay with us.

1.3.2 Context-aware computing

To define it, we could simply say that context-aware computing represents computers' need of getting contextual information that is mandatory to perform their tasks transparently without the need to bother users. Also, the information on the user should be *experienced* without even questioning: just by observing attitudes, actions realised, keeping a track of all this data and analysing it to serve optimally each specific user. A good example of this concept's application is speech recognition. Indeed, most speech recognition software nowadays follows the same installation and set-up pattern: first learn to recognize user's voice to adapt specifically to his usual tone of voice. Here again we could quote many more examples, such as BonziBuddy, a so-called agent, the purpose of which is to assist each user by providing him with accurate information designed for him and nobody else. This shows that the new tendency is towards bringing the information to the user instead of the user having to go for it.

1.3.3 Mobile Agents Paradigm

Mobile agents are a new paradigm other than client/server well-known architecture in which small programs can move from clients to clients on chiefs, interrupting themselves and resuming where they stopped on the previous machine. Roaming agents can collect data transparently to the user, and also mobile agents can be gifted with some advanced artificial intelligence to gain experience from user's behaviours.

In reality mobile agents is a quite young technology that has only existed for some years. Though, more and more programming environments are designed to allow programmers designing mobile agents in an easy manner. There is no specific programming language to design mobile agents, though Java, as a network-oriented language, is particularly well chosen to design agents.

At this point, several pioneers have provided with their version of a programming environment for mobile agents with java. Among them, IBM and their Aglets [6] is probably one of the very most famous examples, we could also quote some universities' projects such as Tracy conceived in a German university, Ajanta and many others. Despite this diversity of choice, the lack of standards and the obvious openness of interpretations of mobile agents' paradigm drive to systems that are far from being perfect right now, and pioneer developers of mobile agents experience important problems. However, this stake is only due to the necessary time to bridge the foundations and standards of mobile agents, after what the paradigm is to become a common developing scheme.

It is interesting to notice that mobile agents can even fit in the well accepted CORBA model or Microsoft's DCOM. To review the previous example of Internet, mobile agents can act as roaming applications going from servers to servers to gather information. As a result, the user will not know: the actual location of the information and even the very existence of agents. Everything looks the same as if it was locally present on the computer, that is exactly what ubiquitous computing is about. On a programming point of view, the application is more reliable than it would with a client/server architecture, here as agents work on a higher level, they are able to adapt more easily to the context; in the case of a wireless network agents are particularly well equipped to observe the evolution of the network, any sudden connection/disconnection of a roaming user. This leads to a better routing of the information; one can then easily understand the role that mobile agents might play in the emerging diversified wireless networks (Bluetooth, Wireless-LAN, and so on). Mobile agents usually need some kind of server applications running on the clients to be able to move from a computer to another, transfers their code but also their general context: to know where they last stopped, their content every single bit of data collected throughout their lives.

Obviously, agents imply a good deal of security issues that have partially been dealt with so far. This is one of the major reasons why commercial application including mobile agents have not come out yet, waiting for some guaranty of reliability of agents to be sure that they are controllable but also that no one outside the radius of concerned people can interfere with them. This last point is undermined on wireless networks, which as a new technology suffer from many security breakthroughs.

2 Overview of mobile agents

2.1 Introduction

This chapter outlines some of the main parameters used in agent-based systems, and highlights the operation of mobile agents. A key worry in any mobile agent system is security, thus a section of this chapter highlights some of the main issues related to security and mobile agents.

2.2 Agent-based systems

Agent is a little piece of program, part of wider application, and the purpose of which is to assist users to do some specific tasks. Generally, White [25] defines that an agent should encompass several basic properties:

- Reactivity to the environment. With this, the agent must be able to react to changes in its environment, such as changes of user, changes of network connections, and so on.
- Autonomy. A key element of agent-based systems is that agents can work without the requirement of a central server. This allows them to operate independent, and even when there is a lack of a network connection.
- Pro-active and goal-oriented. A key factor with mobile agents is that they typically have a single task to complete, such as monitoring for a user login, monitoring user behaviour, and so on. This gives the agent a small foot-print, and also make them easier to test, and are thus more robust.
- Temporal continuity. It is important that an agent should be able to stop its execution, and start it again from the same point that it left off from.

Optionally, an agent can also have the following properties:

- Be communicative. Many systems are now being design using multiple agents, thus agents must be able to communicate with others, and, possibly, with a server.
- Learn from experience. It is often not possible to upgrade agents on a regular basis once they are deployed, thus, as much as possible, an agent should try and learn from its own experience.
- Be mobile. This is the property that mobile agents used, as apposed to a stationary agent which cannot move from one host to another. A mobile agent can autonomously move from host to host, and access local resources through communication channels with stationary agents.

2.3 Main advantages

As with intelligent agents, mobile agents can *learn*. They can thus gain experience from various contexts and adapt their behaviour consequently. More specifically, mobile agents imply numerous benefits [28]:

- They reduce the network load. As mentioned in the Chapter 1, client/server architecture often uses a lot of bandwidth resources, especially when accessing servers, which tend to become bottlenecks. As large volumes are now stored on servers, it is more efficient to move the process to the location of data, instead of moving the data itself. Nayler et al [28] defines that mobility offers an evolution of the object-oriented paradigm where an object can be given a location.
- Agents can operate directly on client nodes thus avoiding network latency while operating locally. This is most relevant in real-time systems where network latency is a huge issue. Braun [33] shows experimental work that in some cases the latency is reduced. This is especially relevant when the transmitted data is relatively small compared with the overall communication time.
- They can embed some protocols, thus allowing for easier updates. A good example of this would be in an update to an application program which uses mobile agents. With this all the application agents could be recalled to a central source and then upgrade, and re-deployed.
- They interact with their environment and adapt themselves.
- They move autonomously. This autonomy along with platform and system independence make them ideal for building reliable and robust distributed applications, and can thus deal with the environment reacting dynamically to changes.

Mobile agents have advantages that make them very useful in a variety of applications such as network administration [28], e-commerce, data warehouses, and so on.

2.4 Shortcomings and limits of agents

Although the concept of mobile agents is over a decade old, the technology proves not to be perfect yet. Many researchers are now developing methods for improving the technology, with more standardization, better programming environments, as well as design patterns that may allow mobile agents to be used in products. In spite of this immaturity of current products, several large IT companies, such as Sun Microsystems and IBM are among the many pioneers of a domain and are researching applications and environments for mobile agents. Before their widespread development, it is indispensable that such a technology gets reliable enough so that it does not stay a research subject and becomes an economical reality. It is obvious that the more an application gets intelligent, the more it also gets unpredictable, dangerous and uncontrollable. That is why most of the current research is focused on security, and also on trying to find a good trade-off between the power of services and the controllability of the agents. Java-based systems already work on tackling some of these security questions by limiting agents' functions to prevent too much damages to client systems or machines. Agimont [2] presents a protection scheme for mobile agents with Java. This study highlights a number of Internet websites relating to these problems of security and system architecture. Already, developers have started to develop agents, especially on the ad-hoc wireless networks.

2.4.1 Agents and security issues

Security is a factor that needs the complete focus of any developers. The possible problems of an agent-based system are:

- Mobile agents damaging local systems.
- Local systems altering or destroying agents. In particular, the danger of embedded viruses is expressed in [3].
- Local systems capturing agents and getting private information.

To prevent mobile agents damaging local systems, Java has a solution by restricting the access to local resources to the static agents. For the problem of mobile agents being hacked or tampered by hosts they cross, the issue is more complicated. It will be up to the programmer to have clean code to limit risks to the minimum. The use of cryptography, for example, can be a good idea for protecting secret information.

Unfortunately, developers will have to wait for the evolution of mobile agents environments. A fully developed and secure system will allow agent technologies which are more trust for businesses.

2.5 Wireless LANs

A wireless LAN is a radio connected local network. The technology has the advantage to make cheap and easy computer connections. Also, users may be able to move wherever they want with their laptops or handheld devices and easily join a network group, that may not be possible for wired networks, which are typically much more complicated to set up and administrate.

2.5.1 Different Wireless technologies

Numerous different wireless technologies are available on the market; some of them are competing on the same level whereas others are complementary. We can distinguish wireless networks, that mainly give computer-like connectivity, from technologies such as the Bluetooth, which aims at linking all sorts of computer-embedded products, such as mobile phones, hi-fi, and so on. Our study focus mainly on wireless Networks, as defined and referenced under the IEEE 802.11 specifications.

2.5.2 IEEE 802.11

IEEE 802.11 refers to a family of specifications developed by the IEEE standardisation community for wireless LAN technology. Accepted in 1997, IEEE 802.11 specifies an over-the-air interface between a wireless client and a base station or between two wireless clients. In fact, 802.11 family encompasses several different specifications:

- 802.11, which applies to wireless LANs and provides 1 or 2 Mbps transmission in the 2.4 GHz band.

- 802.11a, an extension to 802.11 that applies to wireless LANs and provides up to 54 Mbps in the 5GHz band.
- 802.11b, which is also referred to as 802.11 High Rate or Wi-fi. This is an extension to 802.11 and applies to wireless LANs operating at a maximum of 11 Mbps transmission (with a fallback to 5.5, 2 and 1 Mbps) in the 2.4 GHz band.
- 802.11g, which applies to wireless LANs and provides 20+ Mbps in the 2.4 GHz band.

The standard also includes an encryption method, called Wired Privacy Algorithm.

2.6 What mobile agents can offer to wireless

Intrinsically, wireless networks are dynamic, where nodes on the network frequently connect to and disconnect from it. Normal connection algorithms would plainly struggle to deal with these changes efficiently. This is why mobile agents appear to be a key to solve the problem. Agents act like independent individuals that can go around the network from one end to the other, and update their routing table according to the connections that they have made on their travels.

Documents [8] to [15] deal with routing and wireless networks. The concept of mobile agent is intrinsically linked to these issues and often proposed as a solution to adapt existing routing algorithms. Indeed, common distance-vector routing algorithms prove not to be efficient where wireless networks are continually changing. This is why a lot of research is being undertaken on adapting these algorithms, and make them more flexible. This is the case of the work at MIT Media Labs. Their aim is to decentralize the routing by dispatching agents travelling through the network. This, they hope, will make networks more scaleable. On the other hand, Boppana [8] highlights the issue of the overhead spawned by mobile agents; wondering whether they may become more expensive in terms of bandwidth, as routing information constantly crosses the span the network.

Java is currently the most popular language to design mobile agents, for its well known qualities: openness, flexibility, advanced networking. Chapter 3 will now present practical mobile agent programming.

3 Programming Mobile Agents

3.1 Introduction

A key element of mobile agents is the development system, which should provide a platform for the services required for mobile agents. Java is very often used as a programming language as it has enhanced networking features [4], has good multi-platform support, and supports a robust security platform. Before we go deeper in the subject of agent programming, it is necessary to explain some basic concepts and models that are related to it.

3.2 Generalities on network programming

3.2.1 Client-Server paradigm

The client-server paradigm is currently the most used one on the Internet. It basically consists in a server, which advertises one or several services, possibly accessing resources it knows how to access (this is known as "*known how*" capability). The server executes one or more services requested by clients (this is known as "*processor*" capability). The client-server paradigm is used by a wide range of technologies, among them are remote procedure calling, object request brokers (CORBA), and Java remote method invocation (RMI). CORBA and RMI will be covered in more detail, later in this chapter.

A major issue is that servers tend to become sources of bottlenecks. Indeed, in connection-oriented client-server systems, such as TCP connections, clients need to wait for server to reply to their original requests. When a saturated server has to deal with too many of these, simultaneously, it will normally queue clients requests until it has processed all previous requests. The clients, in the meantime, may be blocked in an idle state. This client-server method, although easy to implement, and well supported, is obviously not ideal. The emergence of mobile agents, may provide an improved methods of sharing of resources, as processing migrates towards resources.

Java various technologies are appreciated by developers of networking application and distributed computing-relating projects. This is mainly because Java supports a Virtual Machine (VM), portable secure byte codes and developer-friendly language semantics. All these features make Java a very efficient language, powerful, fast, and well applied to the Internet and networking problems.

In a computing world where networks are becoming more and more heterogeneous as well on hardware configurations as on platforms, software or operating systems, Java offers different solutions adapted to programmers' specific needs. The following section describes the main network programming models available for Java developers.

3.3 Java network programming models

Before introducing JAVA features, we need to explain the concept of *remote object distribution*. As mentioned earlier, network applications generally make use of the client-server paradigm. Remote object distribution consists in providing the developers with the ability to make remote calls to external methods. On a network, it is possible to access objects, in the common sense of the term in object-oriented programming. Instead of retrieving the objects, calls are made remotely, and objects are used as if they existed locally. For users this process is totally transparent, which is location transparency, access transparency, and so on.

An important illustration of this is in CORBA (Common Object Request Broker Architecture). CORBA was developed by the Object Management Group (OMG), a consortium created for enabling object components written by different vendors to properly interoperate together on heterogeneous networks. It allows developers to use objects, which can invoke one another remotely, without the need of knowing the actual location of the objects they call, or the language they were implemented in. Moreover, CORBA may be used on any networking configuration, such as in client-server, peer-to-peer, and so on. The aim is to reduce the size of the application codes, thus a new form of computer organization is needed, based on robust applications that use file servers, that are accessed by many workstations. CORBA acts as a manager that deals with applications requests, and allocates resources so that clients access the information more easily.

3.3.1 CORBA

As defined by the OMG, four major parts compose CORBA: the ORB, object services, common facilities, and application objects. The OMG designed an Interface Definition Language (IDL) to interface between CORBA objects, advertising an object by communicating its methods and parameters to others.

Normally, the implementation is based on two kinds of components: stubs and skeletons. These are responsible for making the location remote methods transparent. Applications are then able to use them the same way they would access local methods. Stubs and skeletons communicate together using a communication manager. Figure 1 show stub and skeletons interact on client-server architecture.

A mapping from IDL is needed for all CORBA objects to the programming language that was used for the implementation and that is what generates stubs, the compiled IDL for the objects. Objects can then be broken, as stubs encompass all detailed signal processing and exception handling needed to operate properly.

The Object Request Broker operates between the data and the application layer. Object Services, represent domain-independent interfaces, which advertise services generally either based on names (Naming Service) or on properties (Trading Service). Common facilities are applications that are user-oriented. Domain interfaces are specific to domains of the applications. Application Interfaces are designed for specific applications, and implies that they are not standardized so far.

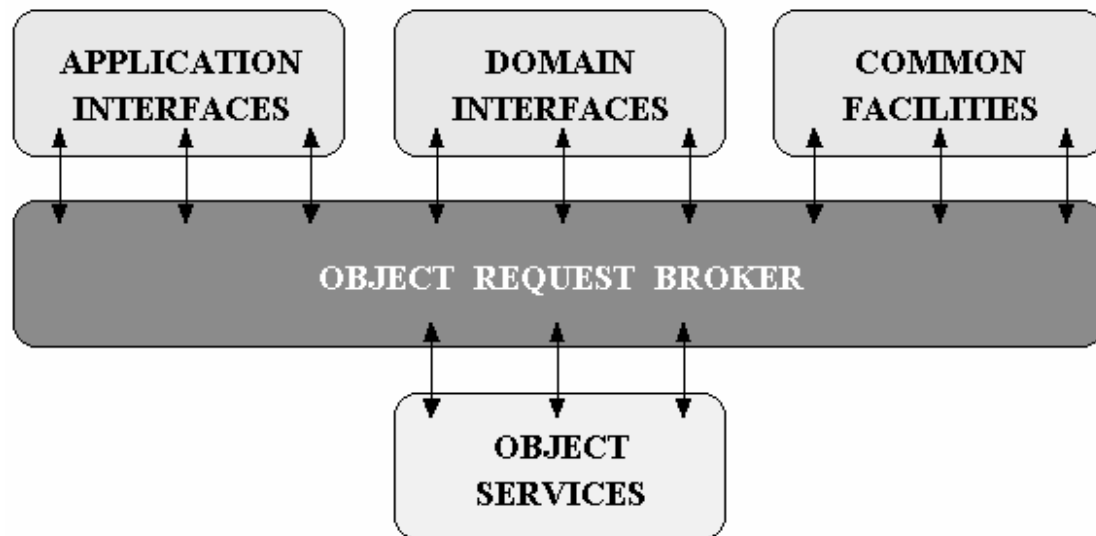


Figure 1: *OMG Reference Model Architecture for CORBA [20]*

3.3.2 RMI

Java RMI (Remote Method Invocation) is a distributed object model designed for the Java Platform. Contrary to CORBA, RMI is unique in that it is centred on the Java language. RMI extends the Java object model beyond the context of a single virtual machine address space. Object methods can be invoked between different Virtual Machines across a network, and actual objects can be passed as arguments and return values during method invocation. Java RMI uses object serialization to convert object structures to byte-streams for transport. Any kind of Java object type can be passed during invocation, including primitive types, standard or user-defined classes, and Enterprise JavaBeans (EJB). The ability to pass actual objects constitutes an adaptation of procedural RPC (*Remote Procedure Call*, see above), for the object-oriented model. This method of serialisation of objects, which could be a mobile agent program, and the ability to move execution from one host to another is especially important for mobile agent systems.

As for CORBA, clients interface with the RMI through a stub, whereas servers interface with RMI through skeletons. Stubs and skeletons are program-specific, and require to be generated. Fortunately, the latest versions of the Java Development Kit (JDK) include the utility called `rmic` to generate stubs and skeletons for any given class or set of classes a developer might write via their interface. At the execution, RMI transparently loads both stubs and skeletons so that client(s) and server(s) can communicate together. A directory service is used so that the platforms hosting the client and server can know where to find the stub and skeleton.

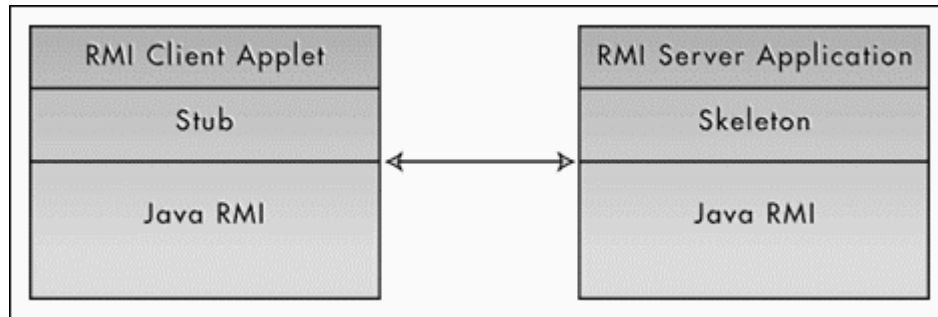


Figure 2: RMI model representation[36]

As Java RMI can dynamically resolve method invocations across VM boundaries, it provides a fully object-oriented (OO) distributed environment. Developers can simply use OO design patterns in distributed computing in the way they would in local programming. RMI is a good alternative solution as it allows developers to use a single model (the Java model) instead of having to mix with differing technologies (CORBA, IDL, and so on). The underlying system of RMI also provides distributed garbage collection. Although CORBA benefits for a greater degree of standardization and a wide acceptance on the market, RMI can be very valuable in an *all Java* application, which explains, why Sun (and JavaSoft) still carry on developing RMI, as well as CORBA. More than that, RMI exhibits its own distinctive features:

- Java embodies robustness, portability, security features, and relatively easy language semantics code on client machines. The task of application distribution is so very much simplified.
- Sun promised a certain number of future enhancements and additional services which will be added to the Java RMI environment. In the JDK (1.2), support for remote object activation (enabling persistent remote references) and custom socket types (enabling SSL encrypted transport) has been performed. Activation enables persistent remote references to objects without requiring that those objects remain executing in a runtime environment at all times. This feature is important for large-scale distributed systems where memory use is preponderant. Also, as much as custom socket types are concerned, a safe encrypted transport of data using SSL (Secure Socket Layer) encryption algorithms is available. Another addition is the support for multicast remote method invocation, server replication strategies for failure protection, and customisable remote reference types.

3.4 Agent Platform evaluation and comparison

A large number of competing agent programming platforms are available on the WWW with varied features. For each application, the choice of the platform is different. A comparative study of the solutions is thus necessary. Solely for Java, several different agent-programming platforms are available. The focus here is on three main products that appear to be most popular choices and represent the more elaborated achievements at this time:

- Aglets designed by IBM.

- Grasshopper [7] which is developed by a less famous German company named IKV.
- Tracy who was designed by a German University and on which an Honours project was focused at Napier last year.

Nguyen et al. [5] contrasted nice of the major environments.

3.4.1 IBM Aglets

The Aglets Software Development Kit (ASDK) is an environment for programming mobile agents in Java. Aglets are the name given to Java objects that move autonomously from host to host. An aglet is able to execute, halt its execution on a host, dispatch itself to another host, and resume the execution there. Aglet supports strong mobility, which means that program code, as well as data, move with the agent. In weak mobility the current state of the program is lost.

The current version of ASDK (2.0.2) is freely distributed. Its structure is relatively clear and simple, and the provided GUI, known as Tahiti server works well. The documentation is of good quality. More important, MASIF standard (designed by OMG to standardize mobile agents environments and let them interoperate) is implemented, as well as support for CORBA.

With communication, sockets, message passing, ATP may be used. Security has been taken into consideration, and the software provides with security mechanism from the User Interface. Unfortunately, Aglets are far from being perfect, the product suffers from a lack of features, more it does not work properly on certain operating systems, and it does not currently offer support for PDAs environments. The research work at Napier is now focusing on hand-help devices to route data, thus the aglets environment may not be the best for this research.

3.4.2 Grasshopper

Grasshopper is currently in its 2.2.3 version. The binary is free, and available from the official website. The GUI is easy to use, and the documentation includes both the theory and practice of mobile agents. It supports many platforms, including Windows CE and even a WWW plug-in for use with Java Server Pages (JSP). Standards MASIF, FIFA are implemented, and like Aglets it works with CORBA.

Communication may be synchronous, asynchronous, dynamic, multicast, with the ability of using different transport protocols, such as sockets, RMI and IIOP. The mobility is weak, but strong mobility can be simulated. Also, the security policy includes confidentiality, data integrity and mutual authentication.

3.4.3 Tracy

Tracy is not distributed over the Internet, but can be requested to the University of Jena in Germany for a purpose of research. The documentation is not available on the Internet either, but a PhD thesis by Peter Braun gives advanced theoretical pieces of information. The next chapter focus on Tracy as the environment was part of an Honours Project in Napier last year.

3.5 Agent-based system architecture

The agent architecture used is based on Rietztler's model [35]. His work was mainly focused on Tracy environment. Tracy is a mobile agent system developed at German Friedrich Schiller University, Jena. This environment, entirely programmed in Java, embodies a lot of very interesting features and boasts particularly advanced migration features compared to its competitors. One of the peculiarities of Tracy is that it offers a synchronous communication mode, based on message passing, but also an asynchronous one with message posting and retrieving on a blackboard. All these techniques having been studied in his work.

The architecture uses mobile agents which interact with static agents which interface with databases. There are three principal elements:

- **Mobile agents.** These were created at the user-end, and wait for SQL requests, before distributing them to a set of agent servers, and then finally going back to each of them to retrieve results for the initial query.
- **Gateway agents.** These were created on all the servers, receiving messages that contain SQL requests, executing them against a local database, and finally passing the result on to the *blackboard* (as defined in Tracy) for a mobile agent to come and retrieve it back.
- **Database agent.** This is an agent at the user-end allowing clients to enter SQL queries for mobile agents, and finally format and display results on the console.

For all database operations, JDBC and MS Access have been used. Thomas Rietztler also highlighted one of the shortcomings of Tracy in that it does only support message passing of strings and not of classes, which is not very adequate for programming clean oriented object applications. In spite of this interesting study, the Grasshopper environment was finally chosen, mainly because of a real contrast in the amount of support between the two products. It also supports a Window CE platform which will allow mobile agents to run on Pocket PC-type devices.

3.6 Selected environment

The choice of Grasshopper has been determined after an overview of different competing products, in regard of the project's needs. Grasshopper's first quality is that it can be used on a variety of operating systems, including Unix, Windows 9X and CE contrary to IBM's aglets, which is more tied to specific operating systems. It is so possible to install the Grasshopper on PDAs, which has been verified at Napier with its installation on Pocket PCs. This proves to be very relevant for a project based on wireless networks. The latter are indeed fitted for heterogeneous machines, including handheld devices such as PDA's that, in most cases, have wireless connectivity. In fact, several scientists, some among them published papers on the Internet, have already made some research on Mobile Agents over Wireless Networks with the Grasshopper environment. In spite of that, the technology is still not mature and research remains more than necessary.

What is more, the provided documentation for Grasshopper is excellent. Even a newsgroup forum exists on the WWW site, which makes it very easy to get advice

from other developers using the platform. Finally, language semantics are the basic ones from Java, what makes it quite easy and straightforward to program Grasshopper Agents.

3.7 Grasshopper environment

3.7.1 Introduction

Grasshopper uses the MASIF standard, set by the OMG, that allows interoperability between the different environments, as well as other existing standards such as CORBA or RMI. The communication module is not to be underestimated either as it can use synchronous/asynchronous communications, multicast, and dynamic communication, as well as supporting different transport protocols, such as sockets and RMI [5]. This large choice makes Grasshopper very flexible and adaptive. Agents made in Grasshopper are able to execute on one machine, and then move to another and continue their execution to a second machine, and so on.

3.7.2 Grasshopper's Requirements

Grasshopper is provided as a set of Java packages¹ that need to be added to the classpath when compiling any agent. It requires a Java Virtual Machine (JVM), Version 1.2 or later. (See Appendix for how to install Grasshopper)

3.7.3 The architecture of Grasshopper

The organisation of any agent-based system in Grasshopper is based on two types of entities:

- **Agencies**, this is the actual runtime environment for mobile and stationary agents [7]
- **Regions**, this is the location where one or several agencies operate in parallel, allowing for a better representation of the organisation by dispatching agents to different regions for their specific purpose.

Basically, Grasshopper is an extension of Java JDK. It includes classes that need to be added to the path for Java to use them as extra libraries. In these agencies, classes directly deriving from Grasshopper agent classes have to be implemented and programmed depending on the purpose. The **agent** can execute itself by moving from agency to agency, while performing certain operations on its way.

3.7.4 The General User Interface (GUI)

Figure 3 and Figure 4 show the main windows that compose the interface of Grasshopper to manipulate agents.

¹ JARs, which are Java Archives



Figure 3: *Region Explorer Window*

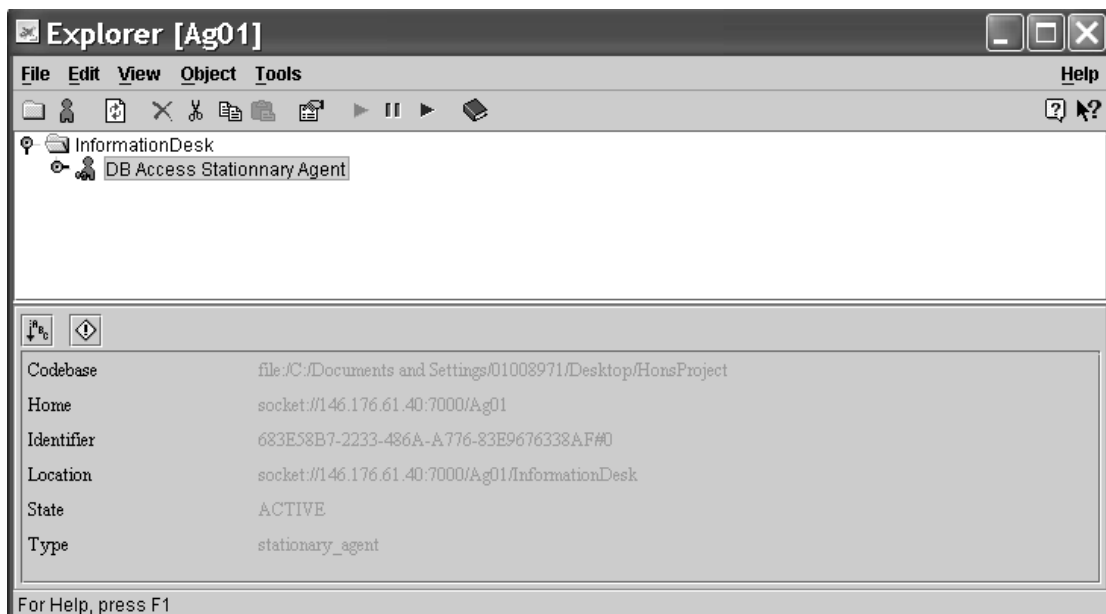


Figure 4: *Agency Explorer Window*

4 Implementation involving Mobile Agents and Wireless

4.1 Aims

This projects aims to investigate mobile agent resources and development systems, along with the investigation of their migration over wireless networks. The research scope integrates the main advancements of the technologies involved, to be able to provide with an improved design for migration that would take benefit from mobile agent migration strategies. Finally, a set of tests procedures will evaluate the performance of the final system, with a special focus on the trade-off between processing and data gathering within agents, to answer the question of to what extent Mobile Agents can be useful. This will highlight the overall performance of mobile agents in ad-hoc wireless networks, their limitations, and provide with some possible applications of mobile agents to ad-hoc, wireless networks.

4.2 Issues involved in the routing on wireless networks

Routing is based on metrics such as the estimation of bandwidth, processing power, network congestion, and so on. Virtually, this information can be obtained from any of the computers on the network. New routing metrics are likely to be used to define a Quality of Service (QoS) for routing over ad-hoc networks. This QoS will guarantee that network traffic operates within a given service requirement. It is thus important the new metrics are used in to define this QoS, and thus the best route. These might involve tests on network latency, processing speed, memory capacity, and so on.

In the Java model, static agents can be run on the system and be granted a high level of security trust on the system, thus a static agent can poll the device for its metrics. A mobile agent, of course, cannot be trusted on the system, as it may be a malicious agent. Thus, mobile agents must communicate with a static agent, in order to determine the system parameters. An authentication mechanism can be built into the communication between the static and the mobile agent, so that both of them authenticate each other. Java has strong authentication methods, as well as standardised public-key encryption methods.

Routing systems make use of agents that have embedded protocols. These have a great deal of responsibly for the efficiency of the whole system. Whereas the research in that area is still in development, current developers of systems designed for wireless networks employ genetic algorithms and complex mathematic calculations to design their routing algorithms.

4.3 Study of the research undertaken in the domain

Different kind of migration algorithms exist:

- Broadcasts: regularly, the state of the network is checked and leads to the updates of all the routing tables of all hosts.
- On-demands: every time a host wants to access a remote resource, it asks its neighbours to propagate its request to find the optimal path at the present time.
- Hybrid, mixing the two previous algorithms.

Up to now, most routing algorithms were based on broadcasting. For static networks, broadcasting works correctly, but according to most recent researchers, on-demand algorithms prove to be more efficient on wireless networks. Routing on wireless networks, indeed, operates best with this kind of algorithm. The dynamism of wireless networks makes it necessary to update routing tables with a much higher frequency rate.

4.4 Design of a model of mobile routing

As discussed in Section 4.3, the mobile routing is based on an on-demand algorithm. The prototype is based on a set of stationary agents running on different Grasshopper environments (and so on different VMs). These static agents simulate the retrieving of metrics data by reading randomly generated values out of a Microsoft Access database.

Once these static agents run on hosts across the network, mobile agents can then be spread to move on any given migration path made of any list of IP addresses provided by clients. Mobile agents will contact static agents on their way, and at the end of their travels they will come back to the initial client from which emanates the request, and display the results (Figure 7 and Figure 8). Migas [35] has outlined the architecture of MARIAN which is a framework for Mobile Agents for Routing, Topology Discovery, and Automatic Network Reconfiguration in Ad-Hoc Wireless Networks, outlined in Figure 5 and Figure 6.

In a fully operational application, data would be analysed with some algorithms locally to determine the best path to be chosen and finally route packets in respect to that choice. The routing itself could be based on: provided the least number of hops on the route; the route with the less electrical power consumption (which is important in hand-held devices); using a certain type of system (such as only PDA's or only workstations); use only nodes in a certain domain; and so on. Our prototype did not aim at dealing with these specific issues but rather focused on the communication of mobile agents, so this was not implemented. The system, though, provide the concept of mobile agents communicating with static agents, which read data from a database.

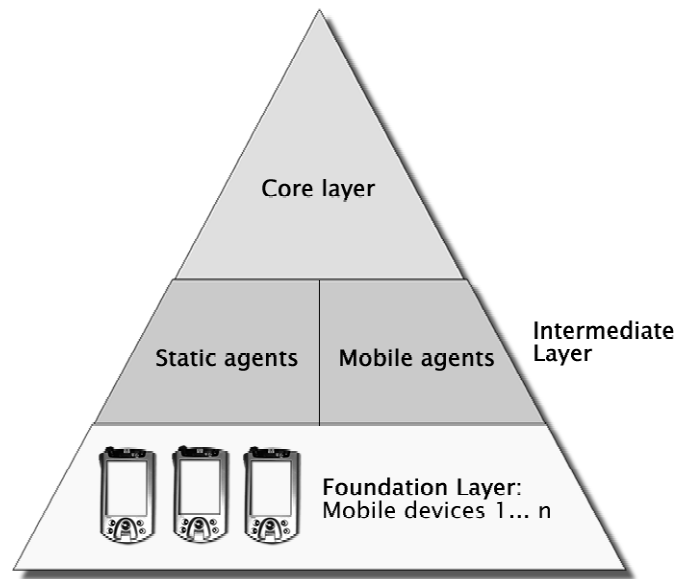


Figure 5: MARIAN layered model for Ad-hoc routing [34]

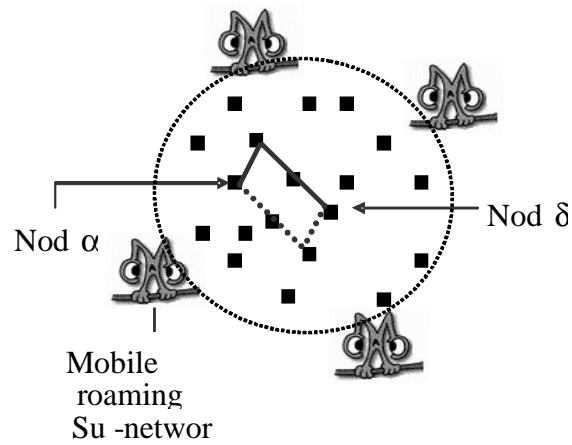


Figure 6: MARIAN layered model using mobile agents [34]

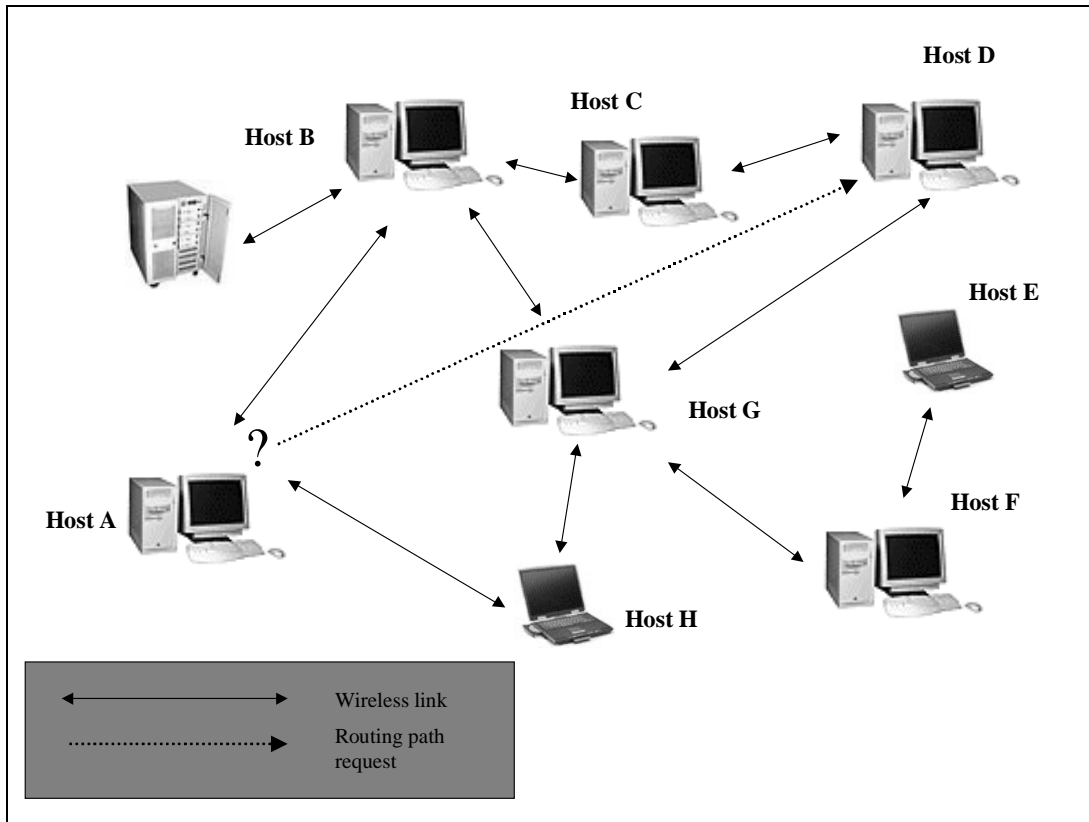


Figure 7: Routing over wireless networks

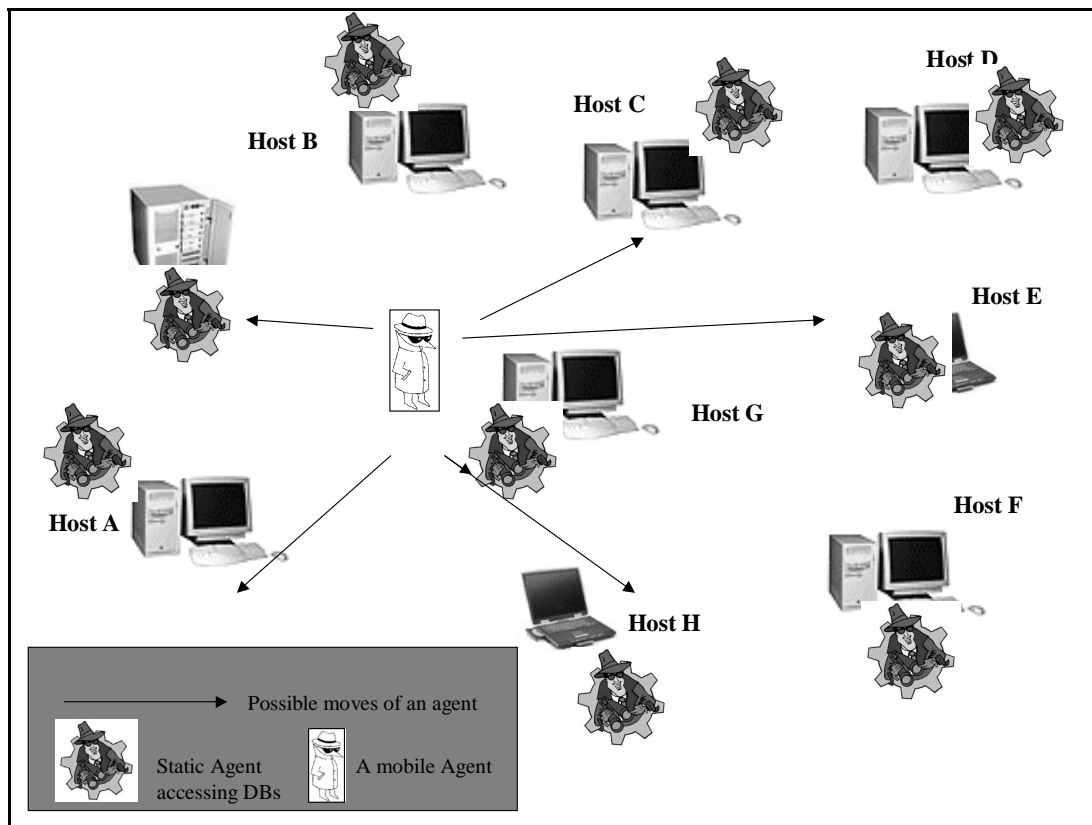


Figure 8: Routing Agents

4.5 Mobile agent migration

The general algorithm for moving the mobile agents from host to host is:

- If it is first host then move to the next hop.
- Save results from local database. This is achieved through communication with local stationary agent which executes a SQL request on the database, and move again.
- If back on the initial host then display results.

The static agent must wait and listen for the mobile agent to arrive at the host. When it does, the mobile agent makes a request to the static, which in turn will read the database, which contains the routing parameters, and passes the data to the mobile agent, which will then migrate onto the next host in the itinerary.

4.6 The process of coding

The following sections give code snippets for the mobile and static agent code. The static agent reads from the database, and will communicate information with the mobile agent. The mobile agent must be able to migrate from one node to the next, get the required data, and move on. The operation of the systems is that initially the static agent is created on each of the hosts. The operation on each host is:

1. Mobile agent arrives at the host, and searches for the static agent.
2. Mobile agent creates a proxy so that it can find the static agent.
3. Once found, the static agent reads the database content.
4. Mobile reads from the proxy and stores the information in a table.
5. Mobile agent leaves the host and carries on its migration.

Finally, when the mobile agent returns to its creator, it displays the results of the visits.

4.6.1 Grasshopper static agent programming

The following class realizes the static agent so that the class is defined as a static agent:

```
public class mobileAg extends  
    de.ikv.grasshopper.agent.StationaryAgent implements testInterface
```

Next, some communications classes are added with:

```
import de.ikv.grasshopper.communication.GrasshopperAddress;
```

Finally there is a `live()` method which contains all the processes of the agent. This is added with:

```
public void live() { // code added here}
```

These are added to the code as follows:

```
*****
import de.ikv.grasshopper.communication.GrasshopperAddress;
import java.sql.*;
/**
 * This class realizes an agent that moves forth and back between
 * two agencies.
 */
public class mobileAg extends
    de.ikv.grasshopper.agent.StationaryAgent implements testInterface
{
}
public void live() {}
*****
```

4.6.2 Read a database: JDBC with Access

JDBC (Java Database Connectivity) is one of the most flexible database systems, and was chosen as it can be used on most types of systems. JDBS is actually a set of classes providing Java Programmers with all the methods to work with SQL databases. The driver for JDBC/ODBC is added with:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
```

The connection to the database is made with:

```
Connection con=DriverManager.getConnection(url);
```

where *url* represents of the address of the database relative to the host. Finally queries on the database are executed with:

```
ResultSet result=select.executeQuery("select field FROM Table");
```

where `ResultSet` contains all the results for the specific query. These are added to the code as follows:

```
*****
try
{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
    System.out.println("here we Are");
    String url="jdbc:odbc:testdb";
    Connection con=DriverManager.getConnection(url);
    Statement select=con.createStatement();
    System.out.println("here we go again");
    ResultSet result=select.executeQuery("select field FROM Table");
    while (result.next()){
        System.out.println("hi");
        String forename=result.getString(1);
        System.out.println("name: "+ forename);
    }//while

}
catch (Exception e){
    e.printStackTrace();
} //catch
*****
```

4.6.3 Grasshopper mobile agent programming

The following class realizes the mobile agent so that the class is defined as a mobile agent:

```
public class TestServerAgent extends de.ikv.grasshopper.agent.MobileAgent {
```

Next, some communications classes are added with:

```
import de.ikv.grasshopper.communication.*;
import de.ikv.grasshopper.agent.MobileAgent.*;
```

Finally there is a `live()` method which contains all the processes of the agent. This is added with:

```
public void live() { // code added here }
```

These are added to the code as follows:

```
*****  
import de.ikv.grasshopper.type.*;  
import de.ikv.grasshopper.communication.ProxyGenerator;  
import de.ikv.grasshopper.agency.IRegion;  
import de.ikv.grasshopper.util.SearchFilter;  
import de.ikv.grasshopper.communication.*;  
import de.ikv.grasshopper.agent.MobileAgent.*;  
import java.util.*;  
import javax.swing.JOptionPane;  
  
public class TestServerAgent extends de.ikv.grasshopper.agent.MobileAgent {
```

4.6.4 Pop up to get IP Addresses

The following code uses the `showInputDialog()` method to create a message box to get users input. In this case, the agent will ask for the destination of its migration:

```
location = JOptionPane.showInputDialog(null, "Where shall I go?");
```

4.6.5 Moving the agent

The first operation of the mobile agent is to move, after which it performs actions. Mobile agents are migrated using a socket connection on port 7000 on the destination host. The `GrasshopperAddress()` class creates a new instance of the connection, over which the agent migrates using the `move()` method. The added code is:

```
move(new  
    GrasshopperAddress("socket://" + location + ":7000/Ag01/InformationDesk"));
```

Where `/Ag01/InformationDesk` is the name of the agency, which is the actual environment where agents live.

4.6.6 Contact local agency and finding static agents

The local agency is the environment where all the agents are able to live, and interact with their local host. It first gets the local region, which contain a number of agencies, with the `getRegion()` method:

```
IRegion regionProxy = getRegion();
```

Next the syntax file searches for a local static agent, so that the mobile agent can communicate with it. The search filter is defined with:

```
SearchFilter filter =
```

```
new SearchFilter(SearchFilter.NAME+"=DB Access Stationary Agent");
```

and with the `listAgents()` method to find all the agents. In this case the search filter will look for agents with the name of DB Access Stationary Agent. There can be more than one static agent, in which case it will take the first one from the search. The search is added to the code as follows:

```
*****  
// Get domain service proxy of local agency  
IRRegion regionProxy = getRegion();  
// Look for the server agent in the  
// agency domain service  
SearchFilter filter =  
    new SearchFilter(SearchFilter.NAME+"=DB Access Stationary Agent");  
serverInfos = regionProxy.listAgents(null, filter);  
// Create proxies of the server agent  
// (One for sync. and one for async. communication)  
  
String serverId = serverInfos[0].getIdentifier().toString();  
*****
```

4.6.7 Create and establish proxies of the server agent

Once the mobile agent has found the static agent, a proxy is used to allow asynchronous communications between the mobile and the static agent. It is used to access remote methods in the same way that would be done locally. The `testInterface.class` interface contains all the signatures of the shared methods. The following adds a new proxy object on which the methods will be performed:

```
testInterface scoutProxy = (testInterface) ProxyGenerator.newInstance  
    (testInterface.class, serverId);
```

This is added to the code as follows:

```
***** // Create proxies of the server agent  
// (One for sync. and one for async. communication)  
  
String serverId = serverInfos[0].getIdentifier().toString();  
testInterface scoutProxy =  
    (testInterface) ProxyGenerator.newInstance (testInterface.class, serverId);  
*****
```

4.6.8 Final actions

The database can be read using the `delaballe()` method. Finally the results for the database can be viewed with the following:

```
String[] tab=scoutProxy.delaballe(); //get results from remote method
for(int i=0;i<tab.length;i++) resultat[i+max]=tab[i];
*****
public void displayRes()
{
    for(int i=0;i<resultat.length;i++) {
        if(resultat[i]!=null)    log("Value "+i+ " = "+resultat[i]);    }
    }
*****
```

4.7 Elements of implementation

4.7.1 The Agency Explorer window

The agency is where mobile agents exist, and the agency explorer allows the user to view and manage the mobile agents which have migrated to the host, and also the static agents. Figure 9 displays a static agent and several mobile agents in an Agency from the view of the agency explorer window. In this case there are four mobile agents, and a single static agent (which is named DB Access Stationnary Agent). These agents are contained within an Information Desk.

The lower part of the screen displays the properties of the agent. In this case, for example, it displays its name, its location, its type (mobile or static) and its state (active or in-active).

4.7.2 The console window

Figure 10 shows the console window at the end of the life of a mobile agent. It displays all the information it gathered all the way through the migration path. It can be seen that the list of values (originally in databases) is displayed first and then followed by the time when the mobile agent stopped its execution. As additional information, the duration of the life of mobile agents is displayed (this will prove very useful for the tests). In this case, the mobile agent displays IP addresses. These are the values that are contained in the database on each host.

4.8 Conclusion

The implementation reached the initial expectation, which is a working version of the Java code that will illustrate the theory of mobile agents. The code itself has been successfully tested without any particular bugs or malfunctions. The system is also scaleable and allowed for doing tests with various configurations: different number of workstations or different databases.

As mentioned it throughout this report, security was a major issue, but still it would be inaccurate to say that the system is foolproof. Nevertheless, this expectation is very

hard to reach, plus it is impossible to reach at this time, as Grasshopper has not been developed enough to take into account extensive security issues. The architecture of mobile and static agents allows for some security, as only static agents have the rights to access the host.

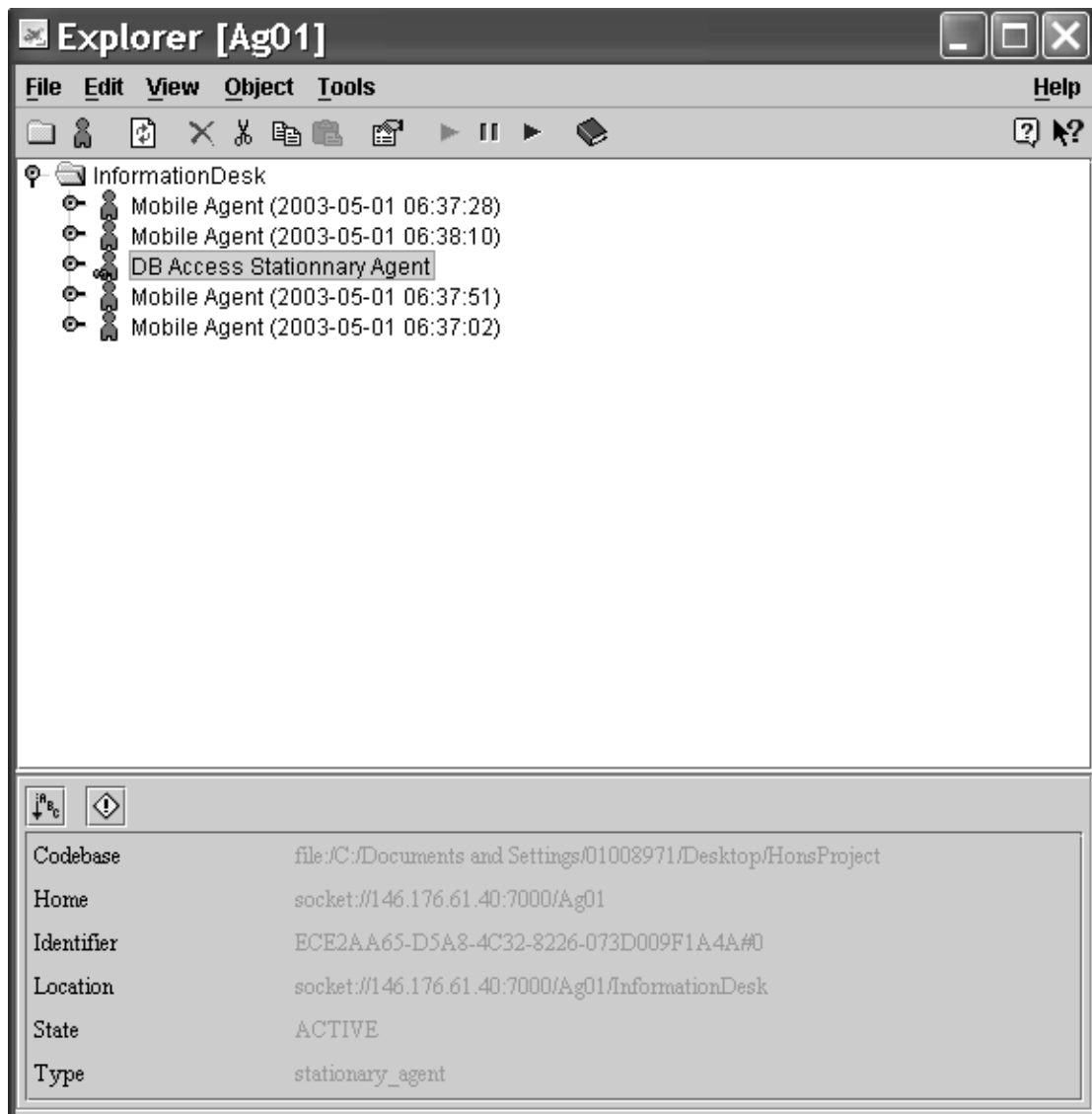


Figure 9: Agency Explorer window

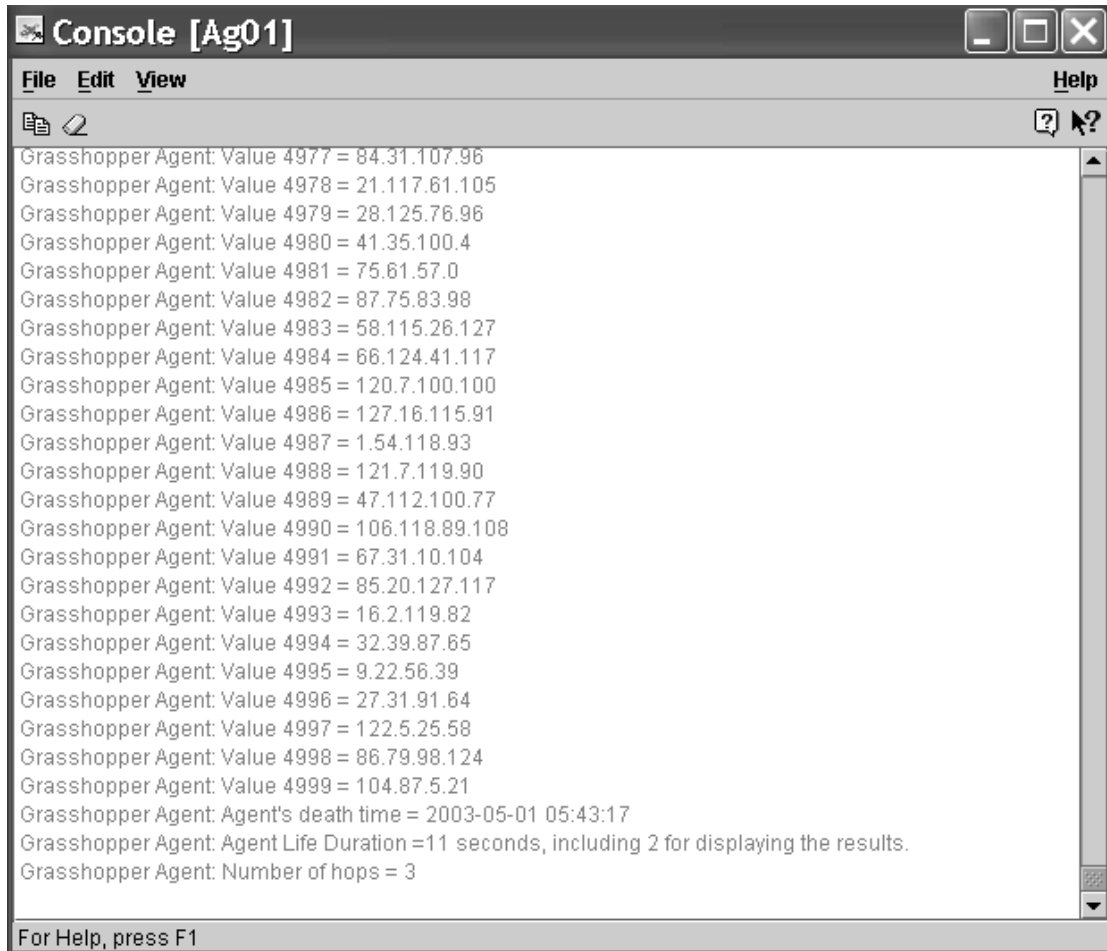


Figure 10: *Console window*

5 Tests

5.1 Introduction

The application is composed of two kinds of agents:

- Static agents. These run on the hosts and effectively access the resources.
- Mobile agents. These move around the network and gathering information by establishing communication channels with static agents we have just mentioned, this on every computer they cross. The information, once retrieved, would finally lead to the effective routing from the initial computer wishing the resources to the location of the one owning them.

The tests aim to show how the size of mobile agents influences the efficiency of the whole system. Indeed, current research shows that the mobile agents' size is an important factor to consider when working with agents.

Business needs require that mobile agents cope with high-scale systems generating a good deal of traffic and data. Our tests will seek to show how relevant this factor is, and whether, or not, some trade-off can be realised between size and speed to use agents only as required.

5.2 Configuration for the tests

The tests were made on a set of wirelessly interconnected computers with no other prerequisite than Java Virtual Machine installed on them, and a mobile agent environment. These tests were first performed in Jack Kilby Computing Centre of Napier, with a number of workstations varying between 1 and 10 and with different databases, the size of which was increasingly important.

To actually evaluate the efficiency of the process with different configurations, timestamps have been used to track the time taken by mobile agents to travel on the network. Unfortunately, the wireless network set by Napier is limited: just a single radio-connection between two computers. This is not sufficient to do any routing of data. The use of PDAs would have helped, and this would provide a future development for the project. Still, the implemented application would work properly on Wireless networks and has been tested successfully on Napier's Wireless LAN.

5.3 Tests results

For the following tests, different executions are represented by dieses (“#”) and values are expressed in seconds. Tests realised in the JKCC labs on four computers connected with a 10 Mbps Ethernet connection. The accuracy of the system is to the nearest second.

5.3.1 Series 1: Migration timing for a four host system with a small database

The first set of experiments uses four hosts (A, B, C and D). A mobile agent starts from the initial host (A), and then migrates across the three other hosts (B, C and D), and finally goes back to the original host (A). Each host has a static agent, which interfaces to a database with 25 values. Thus, the mobile agent will return to the original host (A) with 100 values. The aim of this experiment is to benchmark the mobile agent system with a relatively small payload.

The three tests conducted are:

- Test 1. This involves the normal execution of the application from host A. In this case there is only **one** mobile agent, which is then used for each of the runs.
- Test 2: This is the normal execution of the application from a different host. This is the same as Test 1, and will only have one mobile agent. The only difference is that it starts and ends on a different host (B).
- Test 3: Simultaneous spawning of all the mobile agents. This involves the creation and the sending of all the mobile agents at the same time, and will thus require more resources in this process. In this case a new mobile agent is created for each run, there will thus be **eight** mobile agents in total created for the complete test. As with Test 1, the mobile agent was initially created on Host A.

Table 1 and Figure 11 outline the results. It can be seen that there is an increase in time for Test 3 over Test 1. This is due to the overhead in creating new mobile agents for each of the runs.

	<i>Test 1 (sec)</i>	<i>Test 2 (sec)</i>	<i>Test 3 (sec)</i>
Run 1	7	9	7
Run 2	5	7	6
Run 3	7	8	7
Run 4	6	6	7
Run 5	7	8	5
Run 6	7	7	11
Run 7	5	5	9
Run 8	5	5	9
Average (sec)²	6.2	6.8	7.5
Increase		10.8%	21.6%

Table 1: Results of Series 1 tests

² Average dismisses lowest and highest value.

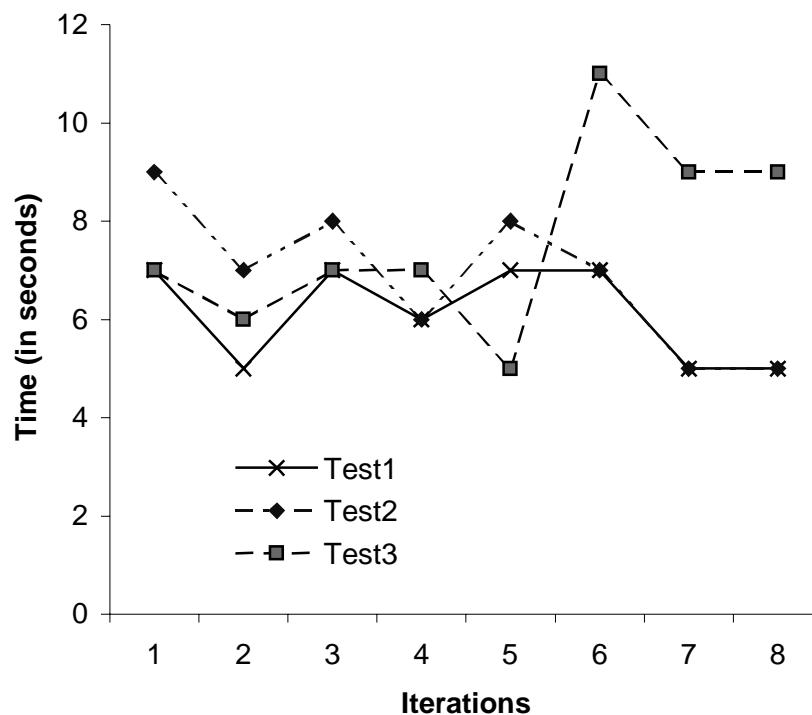


Figure 11: Results of first series

5.3.2 Series 2: Migration timing for a four host system with a large database

The second set of experiments uses four hosts (A, B, C and D). A mobile agent starts from the initial host (A), and then migrates across the three other hosts (B, C and D), and finally goes back to the original host (A). Each host has a static agent, which interfaces to a database with 500 values. Thus, the mobile agent will return to the original host (A) with 2000 values. The aim of this experiment is to investigate how payload affects the performance of the mobile agent system.

As before, the three tests conducted are:

- Test 1. This involves the normal execution of the application from host A. In this case there is only **one** mobile agent, which is then used for each of the runs.
- Test 2: This is the normal execution of the application from a different host. This is the same as Test 1, and will only have one mobile agent. The only difference is that it starts and ends on a different host (B).
- Test 3: Simultaneous spawning of all the mobile agents. This involves the creation and the sending of all the mobile agents at the same time, and will thus require more resources in this process. In this case a new mobile agent is created for each run, there will thus be **eight** mobile agents in total created for the complete test. As with Test 1, the mobile agent was initially created on Host A.

Table 2 and Figure 12 outlines the results.

	<i>Test 1 (sec)</i>	<i>Test 2 (sec)</i>	<i>Test 3 (sec)</i>
Run 1	27	65	10
Run 2	10	72	14
Run 3	14	62	10
Run 4	15	55	14
Run 5	33	44	15
Run 6	13	20	16
Run 7	10	15	12
Run 8	12	16	10
Average (sec)³	15.2	43.7	12.5
Increase		187.9%	-18.6%

Table 2: Results of Series 2 tests

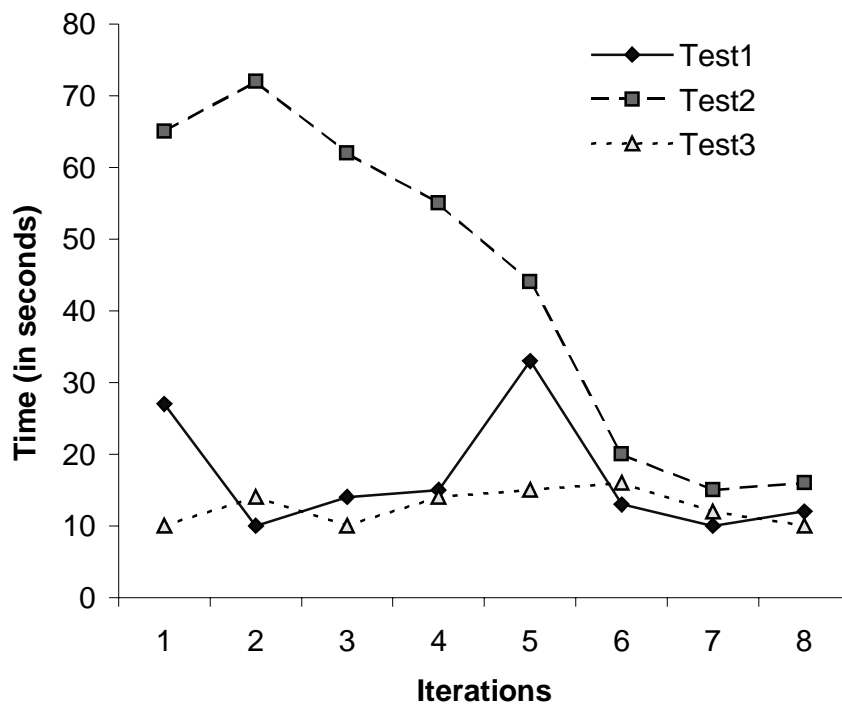


Figure 12: Results of second series

³ Average dismisses lowest and highest value.

5.3.3 Series 3: Migration timing for a four host system with no database

The third set of experiments uses four hosts (A, B, C and D). A mobile agent starts from the initial host (A), and then migrates across the three other hosts (B, C and D), and finally goes back to the original host (A). There is no data transferred between the static agent and the mobile agent, thus the mobile agent migrates without picking up data. The results are shown in Table 3 and Figure 13. It can be seen that in Run 2 there was obviously some disturbance in the network, thus it has been rejected from the average. Figure 14 outlines the overhead of the agents.

	<i>Test 1</i>
Run 1	6
Run 2	86
Run 3	6
Run 4	11
Run 5	6
Run 6	8
Run 7	7
Run 8	6
Average (sec)⁴	6.5

Table 3: Results of Series 2 tests

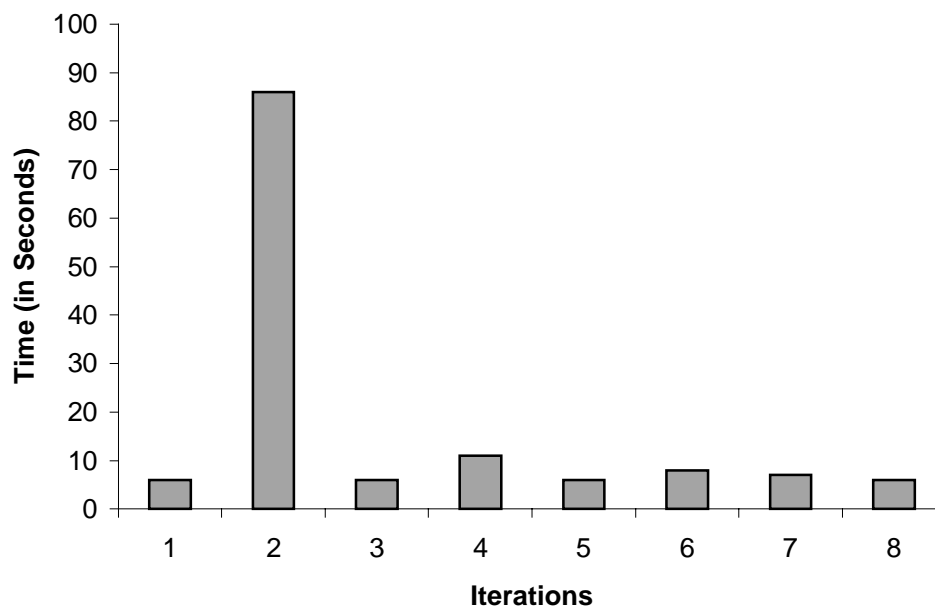


Figure 13: Timing of agents' migration (based on series 3 tests with no database)

⁴ Average dismisses lowest and highest value.

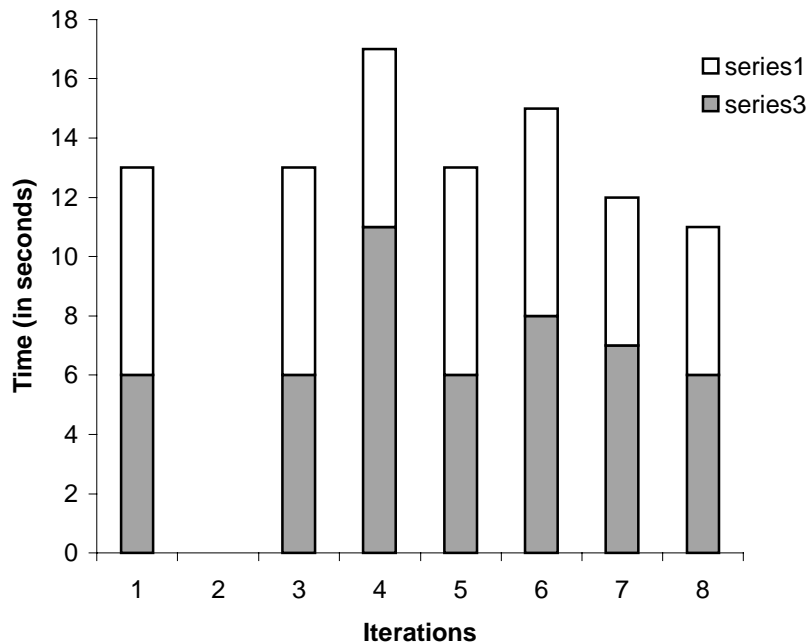


Figure 14: *Overhead of agents (Run 2 has been excluded due to network conditions)*

5.4 Analysis of the results

The third series show the minimum overhead generated by agents when they have no more content than their initial code and some basic state variables. Compared to the third series, there is no remarkable difference on series 2 with a small database. We can judge that the time of process of the database by stationary agents is small. The range of value is very narrow for the first test: between 5 and 7 seconds, which shows that the time of travel of mobile agents is quite regular. The second test shows relatively similar results. The spawning of many agents simultaneously, as shown in test 3, shows that mobile agents operate pretty well. Nevertheless, a small increase for such a small migration (only 4 computers) is still important. This highlights the fact that the environment can easily experience problems of over processing. In the case of large servers that deal with many requests, this can be an important limit, as static agents need to cope with many mobile agents simultaneously. The second series confirm these remarks, and highlights in series 2 that network conditions can change results significantly. Indeed, we can see a large increase of values, and this is surely due to a temporary burden on the network. The last iterations provide with results that are more regular.

Figure 13 highlights the regularity of results but also that agents are prone to drastically slow down in certain contexts, such as temporary network congestion. Figure 14 shows on a same figure both results of series 1 and 3 for the first test, normal execution of the application. This shows that the overhead is regular, usually inferior or equal to the time of process of tasks during the travel.

6 Conclusion

6.1 General

This project proved fascinating, and points towards the next stage in the development of distributed systems. Both wireless networks and mobile agents are still at an early stage just now but there is a good deal of research being conducting in both areas. Agents aim at being users' assistants, and performing simple tasks. Indeed, one of mobile agents' characteristics is their small size, otherwise their mobility is reduced and they generate themselves an overhead on the network. The mobile agent paradigm, although not meant to replace client-server applications in any situation, will probably be complementary.

Mobile agents embody real innovations in comparison with former techniques. Parallel processing increase the speed of gathering information and reduce the problems of bottlenecks.

Regarding security, agent's environments offer the advantage of limiting the access of important resources to static agents. These can filter the information and dispatch a part of it, and so limit the risks of hackers tampering with private data. Nevertheless, the system is not exempt of security breakthroughs, and efforts need to be made to protect agents from being captured by malicious hosts. The technology might require the collaboration of encryption methods, such as Secure Socket Layer (SSL). These issues are especially important on Wireless Networks that are not very secure yet themselves.

6.2 Results

At the end of these tests, we can see what are the main factors conditioning agents efficiency:

- Network conditions, than can change speed of migration, significantly.
- With a larger payload size the migration time of the agent increases as shown in Series 2 tests. Nevertheless, the increase is less than proportional, and may be due to cache effects on hosts.
- As Rietztler [35] observed there may be a caching effect on the migration process, which means that the migration time as the agent re-visits hosts along its travels. This can be observed from Figure 12 where the agent reduces it migration time as it passes through each of the runs. Possibly this may also be due to Java initiating communication sequences, and them keeping them alive, before the agent comes back to visit.

6.3 Future application of mobile agents

This project can give directions to many future works. Wireless networks management based on agents, for instance, could be an idea. One could think for example of analysing a wireless network, with some tools embedded in the code of mobile agents roaming over the network. This could produce a more accurate version than static network management models.

Another idea of work could be the design of applications, the purpose of which is fixing computers locally with mobile agents. A server could store all the tools, and mobile agents could just act as analysers to see possible failures and then recommend the use of any specific tool.

6.4 Further work

The experiments proved the operation of the architecture, and has given ideas of the performance of the system. In the next range of experiments the following could be conducted:

- Increased database sizes. This would involve an increase in the sizes of the databases, as the size of the database is likely to affect the migration times. It is expected that mobile agents will not perform as well if they have large payloads.
- Static agents on PDAs. The static should work well, but it will depend on whether JDBC has been implemented properly on the Java environment, and that the PDA specific drivers are available for database access.
- Mobile agents on PDAs. The Grasshopper environment has been installed on IPAQ Pocket PC's and it works well. In the next range of experiments, the system would be tested on the migration between hosts that included Pocket PC, which run Windows CE. It is expected that it will work well, but it may struggle with large payloads, as these devices typically have limited resources, especially in local memory.
- Power consumption on PDAs. An important factor is the power consumption on mobile devices, thus an experiment can be conducted which monitors the battery consumption in migrating agents through PDAs.
- Processing overhead on PDAs. The process of routing through PDAs should not adversely affect the operation of the PDAs. Thus an experiment is required which monitors the processing and resource overhead when running the static and mobile agents on the devices.
- Mobile agent size. This project has shown that the size of the mobile agent code may have an effect on bandwidth and migration speeds, thus an experiment could be conducted with varying sizes, and functionality, of mobile agents. The aim would be to determine if it is better to increase the size of the static agent, and reduce the size, and functionality, of the mobile agent.

7 Appendices

7.1 Install Grasshopper

The installation of Grasshopper is quite simple. There are different set-up files for different systems (Windows, Windows CE, or Unix). The GH folder on the CD-ROM contains the files which are copied to the local drives.

7.2 Grasshopper settings

Grasshopper can be used with a Windows interface or with typed commands on the DOS console (for Windows systems). The CD-ROM includes examples of batch files to run automatically Agencies and Regions (on which all the environment is based) more easily and faster.

7.3 JDBC Setup on a machine

The use of JDBC [22] is very simple, but still requires one specific operation before being able to work. On the first time, it is necessary to advertise the name of the database to the system. First, one clicks on the *ODBC sources* icon in the configuration panel of Microsoft Windows. After which, a window is displayed and the *Add* button is selected to reference a new database. Once the database driver is selected, and the name of the database specified, everything is operational. See the Figure 15 and 16 for examples.

7.4 Jcreator – Java

This project has used Java JDK 1.4, in collaboration with Grasshopper environment. To effectively program, the Jcreator was used to editing and compile the Java code. JCreator is a free program available on the website download.com (reference). It is very easy to use and has a lesser loading than JBuilder that contains too much features compared to the requirements for this project.

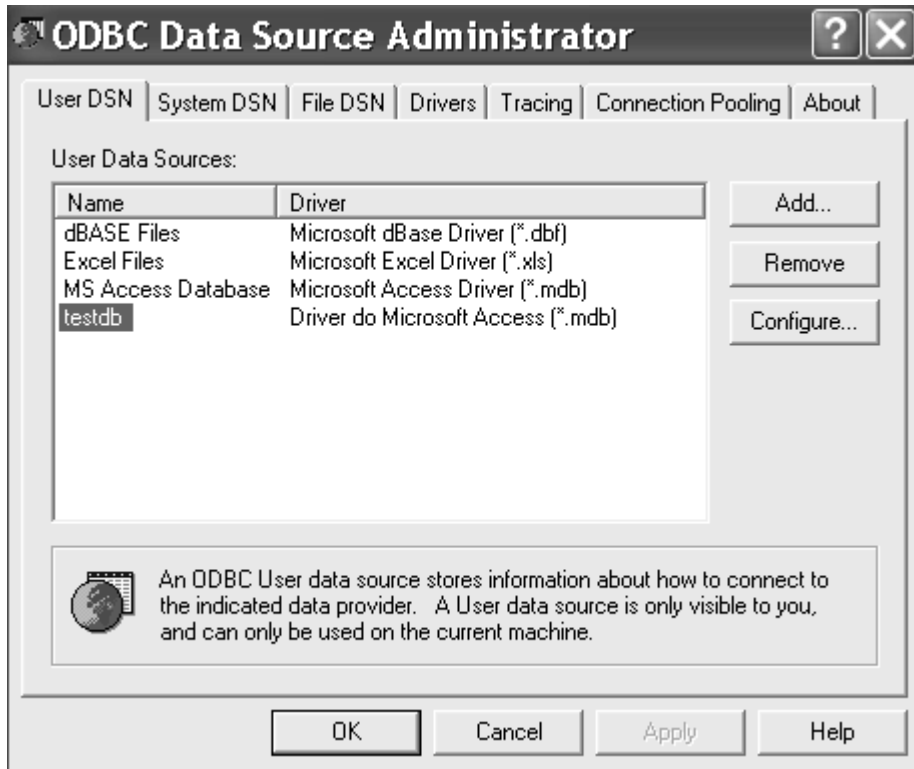


Figure 15: ODBC Data Source Administrator Window

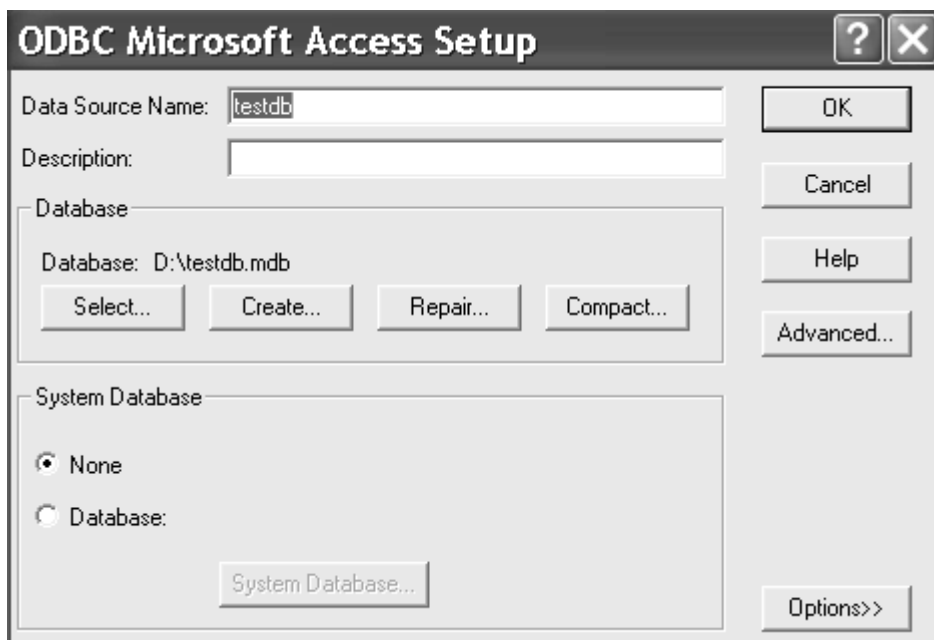


Figure 16: Window to add a database to the system registries

7.5 Static Agent Code

```
*****  
  
// Grasshopper 2.1+  
  
import de.ikv.grasshopper.communication.GrasshopperAddress;  
import java.sql.*;  
  
/*  
 * This class realizes a stationary agent that has access to an Access database.  
 */  
  
public class mobileAg extends de.ikv.grasshopper.agent.StationaryAgent  
    implements testInterface{  
  
    String infos[]=new String[2500];  
  
    public void mobileAg()  
    {  
    }  
    public String getName()  
    {  
        return "DB Access Stationary Agent";  
    }  
  
    public void init(Object[] args) {  
  
    try  
    {  
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();  
    }  
    catch(Exception e){ System.out.println("Problem with JDBC");}  
  
    }  
  
    public boolean imbue (int task) //example method  
    {  
    log(this.getInfo().getIdentifier().toString() +  
        " imbued with task " + Integer.toString(task));  
    return false;  
    }  
  
    public void live()  
    {  
        balle();  
        log("Stationary agent started.");  
    }  
  
    public String[] delaballe()  
    {  
        return infos;  
    }  
    public void balle()  
    {  
        int i=0;  
        try  
        {  
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();  
            String url="jdbc:odbc:testdb";  
            Connection con=DriverManager.getConnection(url);  
            log("Debug Tag 2");  
            Statement select=con.createStatement();  
            ResultSet result=select.executeQuery("select field1 FROM table1");  
  
            while (result.next())  
            {  
                String forename=result.getString(1);  
                log("Value from DB("+i+")"+" : "+ forename);  
                infos[i]=forename;  
            }  
        }  
    }  
    }  
}
```

```
        }
        i++;
    }
}
catch (Exception e){e.printStackTrace();}
}
}
}
*****
```

7.6 Mobile Agent Code

```
*****
// Code written by Alex MANGUER
// With JCreator. Copyright 2003.

import de.ikv.grasshopper.type.*;
import de.ikv.grasshopper.communication.ProxyGenerator;
import de.ikv.grasshopper.agency.IRegion;
import de.ikv.grasshopper.util.SearchFilter;
import de.ikv.grasshopper.communication.*;
import de.ikv.grasshopper.agent.MobileAgent.*;
import java.util.*;
import javax.swing.JOptionPane;
import java.net.*;

/*
 * This class realizes a mobile agent that move around a migration path and
 * retrieves information
 * from stationnary agents based on computers it crosses.
 */

public class TestServerAgent extends de.ikv.grasshopper.agent.MobileAgent {

//initialisation of variables
static boolean firsttime=true;
int creationTime;
String location;
int hop=0; //number of hops
int migration=1;
    //determines how IP addresses of differents hosts are set (manually,pop-
up,by default)
String hostIP[]={"146.176.61.41","146.176.61.36",getlocalIP()};
String[] resultat=new String[2500*hostIP.length];

public String getlocalIP() //gets local ip address
{
    String localIP="";
    try
    {
        InetAddress localaddr = InetAddress.getLocalHost ();
        localIP=localaddr.getHostAddress ();
    }catch(Exception e){}

    return localIP ;
}

public String getTime() //returns the formatted current date and time
{
    /*
    ** on some JDK, the default TimeZone is wrong
    ** we must set the TimeZone manually!!!
    */
}
```

```
    ** Calendar cal = Calendar.getInstance(TimeZone.getTimeZone("EST"));
    */
    Calendar cal = Calendar.getInstance(TimeZone.getDefault());

    String DATE_FORMAT = "yyyy-MM-dd HH:mm:ss";
    java.text.SimpleDateFormat sdf =
        new java.text.SimpleDateFormat(DATE_FORMAT);
    /*
    ** on some JDK, the default TimeZone is wrong
    ** we must set the TimeZone manually!!!
    **     sdf.setTimeZone(TimeZone.getTimeZone("EST"));
    */
    sdf.setTimeZone(TimeZone.getDefault());

    return sdf.format(cal.getTime());
}

public int getTime2() //returns the current date and time as an int
{
    /*
    ** on some JDK, the default TimeZone is wrong
    ** we must set the TimeZone manually!!!
    **     Calendar cal = Calendar.getInstance(TimeZone.getTimeZone("EST"));
    */
    Calendar cal = Calendar.getInstance(TimeZone.getDefault());
    int hour24 = cal.get(Calendar.HOUR_OF_DAY);
    int minute = cal.get(Calendar.MINUTE);
    int second = cal.get(Calendar.SECOND);

    int time=hour24*3600+minute*60+second;
    return time;
}

public int compareTime(int time1, int time2) //gives the difference between
two times, expressed as int values
{
    return Math.abs(time1-time2);
}

public void moveTo()
{
    try
    {
        switch(migration)
        {
            case 1:

                location=hostIP[hop];
                hop++;
                log("hop equals to"+hop);
                if(hop==hostIP.length)
                    firsttime=false;
                break;
            case 2:
                location = JOptionPane.showInputDialog(null, "Where shall I go?");
                break;
            default:
                location="146.176.61.22";
        }

        move(new
GrasshopperAddress("socket://" +location+":7000/Ag01/InformationDesk"));
    }catch(Exception e){}
```

```
}

public void goBackHome() //go back to the initial host
{
    try{
        move(getInfo().getHome());
    }catch(Exception e){}
}

public int resPosition()
{
    int counter=0;
    while(resultat[counter]!=null) counter++;
    return counter;
}

public void arrayConcat (String[] array1,String[] array2)
{
    int max=resPosition();
    for(int i=0;i<max;i++) array1[i+max]=array2[i];
}

public void displayRes()
{
    for(int i=0;i<resultat.length;i++)
    {
        if(resultat[i]!=null)
            log("Value "+i+ " = "+resultat[i]);
    }
}

public String getName()
{
    return "Mobile Agent (" +getTime()+)";
}

public void saveRes() //locates local stationnary agent, and stores the
information it gets from this latter in a table
{
    AgentInfo[] serverInfos;
    // Get domain service proxy of local agency
    IRegion regionProxy = getRegion();

    // Look for the server agent in the
    // agency domain service
    SearchFilter filter = new SearchFilter(SearchFilter.NAME+"=DB Access
Stationary Agent");
    serverInfos = regionProxy.listAgents(null, filter);

    // Create proxies of the server agent
    // (One for sync. and one for async. communication)

    String serverId = serverInfos[0].getIdentifier().toString();

    log(serverId);

    // Establish a proxy to the newly created service agent
    testInterface scoutProxy = (testInterface) ProxyGenerator.newInstance
(testInterface.class, serverId);

    boolean answer;

    // Call a method on the service agent and store response
    int max=resPosition();
    String[] tab=scoutProxy.delaballe(); //get results from remote method
delaballe()
}
```

```
        for(int i=0;i<tab.length;i++)    resultat[i+max]=tab[i];    //store the
database in array tab
    }

public void init(Object args[])
{
    creationTime=getTime2();
    log("Agent's birth time =" +getTime());
}

public void live() //process of the mobile agent from its birth to its death
{
    int beforeDisplayTime=0;

    if(location==null && hop!=hostIP.length)
    {
        moveTo();
    }

    if(firsttime)
    {
        if (migration==1 && hop!=hostIP.length) //save information from the
database
            for all computers on the path, except the initial host
                saveRes();
    }

    if (migration==1 && hop==hostIP.length) //display results on the initial
host
    {
        beforeDisplayTime=getTime2();
        displayRes();
    }

    if (migration!=1)
    {
        goBackHome(); //if only one host is on the path, go back to the
initial host
    }

    if (migration==1 && hop!=hostIP.length+1) //without the last part of the
condition , infinite loop.
        moveTo();

    int DeathTime=getTime2();

    //display some key information on the console
    log("Agent's death time = "+getTime());
    log("Agent Life Duration =" +compareTime(creationTime,DeathTime)+
seconds, including "+compareTime(beforeDisplayTime,DeathTime)+
seconds for displaying the results.");
    log("Number of hops = "+hop);
}
}

*****
```

8 References

- [1] Mobile Code, Agents, and Java Website. <http://www.infosys.tuwien.ac.at/Research/Agents/homepage.html>
- [2] D.Agimont. "A Protection scheme for Mobile Agents on Java". INRIA, France, 1997, <http://citeseer.nj.nec.com/cache/papers/cs/594/http:zSzzSzsirac.imag.frzSzPUBzSz97zSz97-mobicom-PUB.pdf/hagimont97protection.pdf>
- [3] David Chess, Colin Harrison , Aaron Kershenbaum. "Agents are they a good idea". IBM, T.J Watson Research Center, New York, 1995. <http://citeseer.nj.nec.com/cache/papers/cs/1492/http:zSzzSzwww.infosys.tuwien.ac.atzSzResearchzSzAgentszSzarchivezSzspecialzSzmobagtibm.pdf/chess95mobile.pdf>
- [4] Danny B. Lange. "Mobile Agents: Environments, Technologies, and Applications". General Magic, 1998. http://www.iig.uni-freiburg.de/~eymann/avalanche/paam_lange.ppt
- [5] NGUYEN T. Giang, Dange T. Tung. "Comparison of different agent development platforms". Pellucid 5FP IST-2001-34519, June 2002. <http://pellucid.ui.sav.sk/TR-2002-06.pdf>
- [6] IBM Aglets Development Kit Homepage. http://web.media.mit.edu/~stefanm/ibm/AgletsHomePage/index_new3.html
- [7] IKV Grasshopper Homepage, <http://www.grasshopper.de/>
- [8] Rajendra V. Boppana. "An adaptive distance-vector routing algorithm for mobile, ad-hoc wireless networks", Computer Science Division, University of Texas at San Antonio, 2001. <http://citeseer.nj.nec.com/cache/papers/cs/22156/http:zSzzSzwww.ieee-infocom.orgzSz2001zSzpaperzSz603.pdf/boppana01adaptive.pdf>
- [9] John Sum, Hong Shen, Gilbert H. Young and Jie Wu. "Analysis on extended ant routing algorithm for network management", Hong Kong Polytechnic University, Proc. of the Second International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'01), June 2001. <http://citeseer.nj.nec.com/cache/papers/cs/24149/http:zSzzSzwww.cse.fau.eduzSz~jiezSzsumwu1.pdf/analysis-on-extended-ant.pdf>
- [10] Ichiro Satoh, "Application-Specific Routing for Mobile Agents", Proceedings of International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/ Distributed Computing, pp.299-305, the International Association for Computer and Information Series, August, (2001). <http://research.nii.ac.jp/~ichiro/papers/satoh-snpd2001.pdf>
- [11] Hairong Qi .“Optimal Itinerary Analysis For Mobile Agents In Ad Hoc Wireless Sensor Networks”. Electrical and Computer Engineering Department, University Of Tennessee, 2001. <http://www.ee.duke.edu/~vishnus/DARPA/01wireless.pdf>
- [12] Nelson Minar, Kwindla Hultman Kramer, and Pattie Maes. "Cooperating Mobile Agents For Dynamic Network Routing. Software Agents for Future" Communications Systems, Springer-Verlag, 1999, ISBN 3-540-65578-6. <http://xenia.media.mit.edu/~nelson/research/routes-bookchapter/minar.pdf>
- [13] Venkatesh Ramarathinam and Miguel A. Labrador. "Performance Analysis of TCP over Static Ad Hoc Wireless Networks." In Proceedings of the ISCA 15th International

- Conference on Parallel and Distributed Computing Systems (PDCS)", Pages 410-415, September 2002. <http://www.csee.usf.edu/~labrador/papers/pdcs2002.pdf>
- [14] P. Arabshahi, A. Gray, I. Kassabalidis, A. Das, S. Narayanan, M.A. El-Sharkawi, and R.J. Marks II, "Adaptive routing in wireless communication networks using swarm intelligence," Proc. 19th AIAA Int. Communications Satellite Systems Conf., 17-20 April 2001, Toulouse, France.
- [15] Xin Wang, Fei Li, "A Multicast Algorithm Based on Multicast Agents in Ad hoc Networks", IEICE Trans. Commun, Vol E84-B, No 8. August 2007. http://search.ieice.org/2001/pdf/e84-b_8_2087.pdf
- [16] Thomas Rietzler, Mobile agents and Tracy Hons Project Page, http://www.dcs.napier.ac.uk/~bill/pr_2001_2002_05.htm
- [17] Danny B. Lange, "Mobile Objects and Mobile Agents: The Future of Distributed Computing?" <http://www.ifs.uni-linz.ac.at/~ecoop/cd/papers/1445/14450001.pdf>
- [18] Development of CORBA compatible applications, <http://ltiwww.epfl.ch/sJava/version2/CORBA.html>
- [19] CORBA, OMG page, <http://www.corba.org/>
- [20] IEEE 802.11, <http://grouper.ieee.org/groups/802/11/main.html>
- [21] RMI, official website, <http://java.sun.com/products/jdk/rmi/>
- [22] Java Data Base Connection (JDBC, official website), <http://java.sun.com/products/jdbc/>
- [23] Jennings N and Woolridge M, 'Agent Technology: Foundations, Applications and Markets', Springer. 1998.
- [24] Bradshaw J, 'Software Agents', MIT Press, 1997.
- [25] White J, 'Telescript Technology: Mobile Agents', Software Agents, MIT Press, 1997.
- [26] Migas, N, Buchanan W and McCartney K, Mobile Agents for Routing, Topology Discovery, and Automatic Network Reconfiguration in Ad-Hoc Networks, IEEE ECBS 2003, Arizona.
- [27] Hongmei Deng, Wei Li, Agrawal DP. Routing security in wireless ad hoc networks. IEEE Communications Magazine, vol.40, no.10, Oct. 2002, pp.70-5.
- [28] Franklin, Stan and Graesser, Art (1996). Is it an Agent, or just a Program? A taxonomy for Autonomous Agents. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag.
- [29] Naylor M, Buchanan WJ, Scott AV, "Enhancing network management using mobile agents", Proceedings Seventh IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2000). IEEE Comput. Soc. 2000, pp.218-226.
- [30] N.Migas, ECFRC(3) research document, Jan 2003.
- [31] R.R. Chpudhury, S. Bandyopadhyay, and K. Paul, "A distributed mechanism for topology discovery in ad hoc wireless networks using mobile agents", In Proceedings of First Annual Workshop on Mobile Ad Hoc Networking Computing, MobiHOC Mobile Ad Hoc Networking and Computing, August 11, 2000.
- [32] S. Marwaha, C.K. Tham, and D. Spinivasan, "Mobile Agents based Routing Protocol for Mobile Ad Hoc Networks", Symposium on Ad Hoc Wireless Network, National

University of Singapore.

- [33] The Migration Process of Mobile Agents, Peter Braun, PhD thesis, April 2003, University of Jena.
- [34] MARIAN: A Framework using Mobile Agents for Routing, Topology Discovery, and Automatic Network Reconfiguration in Ad-Hoc Wireless Networks, N.Migas, W.Buchanan and K.McArtney, Submitted to Wireless Conference, 2003.
- [35] Thomas Rietztler, Mobile Agents using Tracy, Hons. Project report, June 2002.
- [36] Building RMI Applications,
<http://developer.novell.com/research/devnotes/1998/october/07/04.htm>

9 Gantt

i	Task Name	Duration	Start	Finish	Predecessor	Sep '02	11 Nov '02	23 Dec '02	03 Feb '03	17 Mar '03	28 Apr '03
						S W	S T	M F	T S W	S T	M F
1	Literature Review	35 days	Mon 04/11/02	Fri 20/12/02							
2	First Experimentations on Mobile agents	30 days	Mon 11/11/02	Fri 20/12/02							
3	Project interim Report	14 days	Fri 20/12/02	Wed 08/01/03							
4	Design	22 days	Fri 10/01/03	Mon 10/02/03							
5	Implementation	25 days	Mon 10/02/03	Fri 14/03/03							
6	Tests	14.5 days	Mon 03/03/03	Fri 21/03/03							
7	Analysis	20 days	Mon 24/03/03	Fri 18/04/03							
8	Evaluation of the prototype	10 days	Mon 24/03/03	Fri 04/04/03							
9	Project dissertation	30 days	Tue 25/03/03	Mon 05/05/03							
10	Demonstration/Presentation	1 day	Wed 07/05/03	Wed 07/05/03							
11	Project Viva	1 day?	Thu 22/05/03	Thu 22/05/03							

This table presents the plan of main steps of the project from September 2002 to early May 2003.