1 Introduction

1.1 Introduction and Background

Over the years computer systems have successfully evolved from centralized monolithic computing devices supporting static applications, into client-server environment that allow complex forms of distributed computing. Throughout this evolution limited forms of code mobility have occurred: the earliest being remote job entry (Boggs, 1973) terminals used to submit programs to a central computer and one of the latest being Java applets downloaded from web servers into browsers. In the last few years, a new phase of evolution is underway that takes code mobility one step further by allowing complete mobility of a whole computational component along with its state, the code it needs, and some resources to fulfil its tasks, to a remote site (Fuggetta et al., 1998). Such a system is referred to as mobile agent system with mobile agents as one of its major component.

For the purpose of this dissertation, the definition of mobile agents as provided by Karnik and Tripathi (1998) which states that *a mobile agent is a program that represents a user in a computer network and can migrate autonomously from node to node, to perform some computation on behalf of the user* will be used. A mobile agent system, amongst other things, provides the environment in which an agent operates. Mobile agents, being an emerging technology, have attracted interests from a wide spectrum of research fields. For example agents and agent communication in particular, have attracted attention from the Artificial Intelligence community (Bernakoes et al., 2004). Bettini et al. (2002) have analysed code mobility from programming language point of view, Johansen et al. (1999) from operating systems point of view, while Sycara et al. (2003) look at agents from a software engineering point of view. With all these research activities in various fields, one can easily conclude that the mobile agents must be offering some benefits.

The fundamental reasoning in favour of using a mobile agent in a distributed system is that local method calls are faster than remote procedure call. This is due to the fact that remote procedure calls have to marshal and unmarshal the arguments passed to and the value returned from the remote procedures (Wolfgang, 2000). Additionally, each remote procedure call suffers form the latency of the underlying network. Keeping this in mind, the following conclusions with respect to the usefulness of mobile agents can be drawn:

In a system that is bounded by network speed, mobile agents can help reduce intermediate data transfers by moving across the network to the location where the resources reside.

Mobile agents provide greater flexibility in dealing with network entities and distributed resources in comparison to the client-server paradigm where a server is incapable of acting as a client to other servers (Harrison et al., 1995). Mobile agents are considered to be peer entities and, as such, can adopt whichever communication stance is most suitable to their current needs.

As reported by Harrison et al. (1995), having agents move between nodes gives it the ability to "survive" even when the node that launched it fails. This provides for the improvement in the persistence capability of the system. Furthermore, in a client-server relationship, the state of the transaction is generally spread over the client and the server. In an event of a network or server shutdown during request, it is difficult for the client to reclaim the situation and re-synchronize with the server because the network connection would have been lost. This problem is generally easier to deal with as mobile agents do not need to maintain permanent connections and their states are contained within themselves.

In spite of the numerous benefits obtained by using mobile agents in a distributed environment, some of which are mentioned in the previous paragraphs, using mobile code raises a few serious concerns, paramount amongst them being the security threats to the components of the mobile agent systems. As described in Karjoth et al. (1997), Farmer et al.(1996), and Jansen (2000) these security threats originate not just in malicious agents but in malicious hosts as well.

1.2 The problem statement

In their article, Schoeman and Cloete (2003) have indicated that in recent times, there has been a surge in research activity in mobile agent system arena. This has resulted in an increase in the number of mobile agent systems being developed (Gray et al., 2000) for use in both commercial and research arena. Each of these mobile agent systems differ from one another, to various degrees, with respect to the architectural designs, the implementation languages, the mobility mechanisms, communication related issues, agent management issues and security aspects to name just a few.

As the numerous advantages offered by agents are commonly known (Sierra et al., 2000), one may be inclined to use an existing mobile agent system when building large-scale projects involving agents. Having to choose from over 70 different mobile agent systems (*The Mobile agent List http://mole.infomatic.uni-stuttgard.de/mal/preview.html*) is not an elementary task even for an expert in the field of distributed system as each mobile agent system has its own strengths and weaknesses. For example, one system may have a better migration capability while another may offer superior mechanism for interaction amongst agents and yet another may provide a better solution to the security concerns.

Selecting a suitable mobile agent system from a list such as the above mentioned one is difficult, as it requires an in-depth study of the various properties of the mobile agent systems. To facilitate this process, one may ask the question – what are those properties of a mobile agent system that must be analysed in order to select a desired one for a particular project? To answer this question, a set of factors based on the issues surrounding mobility, communication and security are presented and used for the evaluation of the mobile agent systems. It is aimed that evaluating the mobile agent systems against this set of factors will assist in making an informed decision when choosing a mobile agent system. The reasons for considering only these aspects of mobile agent systems are discussed in the following section.

1.3 Proposed solution

As mentioned earlier, mobile agents have attracted interest from wide ranging fields of research. The reasons being that mobile agent systems have an impact on a wide range of fields of interest to researchers. Considering the vast nature of this field of research, it is not possible to study all aspects related to mobile agent systems. Hence, this dissertation concentrates on mobility, communication and security related issues.

The reasons for restricting to these three issues are as follows:
A mobile agent's primary identifying characteristics, as the name suggests, is the mobility of the agents (Tripathi et al., 2001), (Horvat, 2000) and (Picco, 2001). Without this property, the advantages obtained by using such a system would not warrant the research efforts currently taking place and hence forms one of the aspects to be considered when selecting a mobile agent system.

Probably the most important reason for having the agents to migrate from node to node is to allow them to perform some task on behalf of their users. In order to achieve this goal, the agents need to be able to communicate and coordinate their efforts with one another (Kone et al., 2000). Therefore, for any mobile agent system, suitable communication mechanism between agents must be supported. It is for this reason that agent communication is one of the issues considered for evaluating mobile agent systems (MAS).

According to Farmer et al. (1996), Jansen (2000) and Tripathi et al.(2001), the major obstacle to the widespread acceptance of mobile agents is the security concerns it raises. By virtue of an agent's ability to migrate from host to host increases the threat of security violations, as it makes it possible for an untrustworthy user to dynamic inject malicious agents that can compromise hosting node's resources in a way similar to that of viruses and worms. On the other hand, a host entrusted with the responsibility of agent execution could try to tamper with the agent code and state, with intent to disclose private information and to block agent transfer to successive execution sites (Johansen, 1999). From the discussions thus far, one can easily conclude that security is a major concern in a mobile agent system. For a mobile agent

system to be considered as a serious contender for a development project, it must provide mechanisms to deal with the security concerns adequately. It is for these reasons that agent security has been selected as one of the aspects to consider when selecting a mobile agent system.

From the discussions above, the issues related to mobility, communication and security stands out as being amongst the most important of the issues to be considered when deciding on a particular mobile agent system. It must be noted that these are by no means the only aspects that can be considered when selecting a mobile agent system. Other issues, though not directly influencing the functionality of a mobile agent, but relevant to the environment in which the mobile agent functions, such as the effectiveness of a programming language in developing mobile agents and mobile agent systems could also be considered. Since the focus of this dissertation is at the implementation level, programming language related aspects for the selection of the mobile agent system have not been considered. One could also consider other non-functional related issues such as availability of platform state (alpha, beta or production release) and Rapid Application Development tools, Operating system environment supported (windows/UNIX), support for integrated debugging tool etc.

The following paragraph gives an indication of the factors that will be used for the evaluation of the mobile agent systems.

- The type of mobility supported (Strong / Weak).
- The nature of the itinerary used (Static / Dynamic).
- Is there support for inter-platform messaging?(Yes/No)
- What protocols are used for message transport over the network? (SMTP / TCP/IP / HTTP)
- Are there mechanisms in place to provide security for agents while in transit between various hosts? (Yes / No)

1.4 Strategies for finding a solution

Many mechanisms have been proposed and implemented by mobile agent system developers for the three issues being investigated (Lauvset et al., (2001), Karnik et al. (2001), Jansen (2002), and Gray et al. (2002)). The steps taken to answer the research question were firstly, a critical analysis of the various issues surrounding the three aspects (mobility, communication, security) were performed. From this analysis, sets of factors were obtained. As it is not feasible to evaluate all mobile agent systems, a means of grouping these agent systems are suggested (chapter 6 presents the grouping process). From each group, a mobile agent system was selected and evaluated against the factors obtained. Using this evaluation, a mobile agent system with the desired properties may be selected.

In addition to this, a simple application using the Aglet Software Development Toolkit (ASDK) was developed and referenced throughout the dissertation to explain certain concepts of agent mobility, communication and security.

1.5 Context of Research

Determined by the extensive literature survey conducted regarding the evaluation of mobile agent systems, very few analysis of this nature exist. Altmann et al. (2000) consider issues such as availability (legal and financial aspects for deployment of the platform), documentation (issues for successful use of the agent development platform), and development (i.e. how efficiently programs for the agent platform in question can be designed, implemented, and tested). These factors are external to the mobile agent system. Wittner (1999) has evaluated mobile agent systems from a network fault management perspective. Dikaiakos et al. (2000) evaluate mobile agent systems from performance analysis point of view, whereas information about the agent systems, during their execution, are collected and analysed by Xuhui et al. (2004). The current study uses criteria that are directly related to the implementation of the mobile agent systems and hence are complementary to the ones mentioned above.

1.6 Delimitation of study field

This research is limited to the study of mobility, communication and security mechanisms in mobile agent systems at an implementation level with the objective of developing a set of factors to be applied in evaluating mobile agent systems. The evaluation of the mobile agent systems, against these factors, is limited to just one per group. It should be kept in mind that these are by no means the only assessment criteria that can be used for mobile agents. Other criteria such as availability (of an evaluation version, platform state - alpha, beta or production release), operating system environment supported, how efficient programs for agent platform can be designed, implemented and tested could all be used.

1.7 Structure of the Dissertation

Chapter 1 presents the objectives, the proposed approach to attain the objectives together with the limitations of this research. The organizational structure of the dissertation has also been presented.

Chapter 2 presents an application using Aglets Software Development kit (ASDK), which is used in the discussions and demonstrations of certain issues regarding mobility, communication and security presented in later chapters. Appendix A presents the source code of this application.

Chapter 3 contains information regarding agent mobility mechanisms, code relocation strategies, and data space management. Support for agent persistence and types of agent itinerary have been studied. Evaluating factors for mobile agent systems with respect to mobility have been developed.

Chapter 4 contains communication mechanisms in mobile agent systems together with agent naming and name resolution mechanisms. The need for agent communication

language has been discussed. The factors surrounding agent communication against which the MAS are evaluated is presented.

Chapter 5 presents some of the security concerns and the mechanisms suggested to alleviate those concerns. Evaluation factors for MAS with respect to security have been developed.

Chapter 6 presents the procedure for the selection of the MAS for evaluation against the factors. This chapter also presents the evaluation of the selected MAS for their implemented mechanisms to support mobility, communication and security related issues.

Chapter 7 concludes the study by briefly highlighting some of its findings and suggests direction for future research.

2 A Hands-on Look At Agents

2.1 Introduction

In this chapter a simple application is developed with the aim to demonstrate how agents achieve mobility and communication. The scenario of the experiment is as depicted in figure 1 below. Three PCs namely Jupiter, Venus and Saturn are connected to each other and have the Java virtual machine and the Tahiti aglet server installed on them. Jupiter is the base station on which the agents are created. The stationary agent on Jupiter then creates and dispatches a mobile agent to Venus and Saturn. Any communication from the stationary agent is sent simultaneously to both the mobile agents. Each of the mobile agents can communicate independently with the stationary agent. The stationary agent communicates with the two mobile agents in a synchronous manner whereas the two mobile agents communicate with each other asynchronously. As a means to demonstrate basic security, the application allows for the creation of agents only on one of the hosts - the home base (host from where the agents begin their journey). The discussion on security aspects of this application is presented in chapter 5. It must be further noted that the application does not demonstrate all the features of mobility, communication and security. In the subsequent chapters, this application is used as basic example. Section 2.2 presents a brief description of the agent development environment. The details of each agent and their functioning from a technical perspective are presented in section 2.3. Section 2.4 presents a brief summary of the chapter.

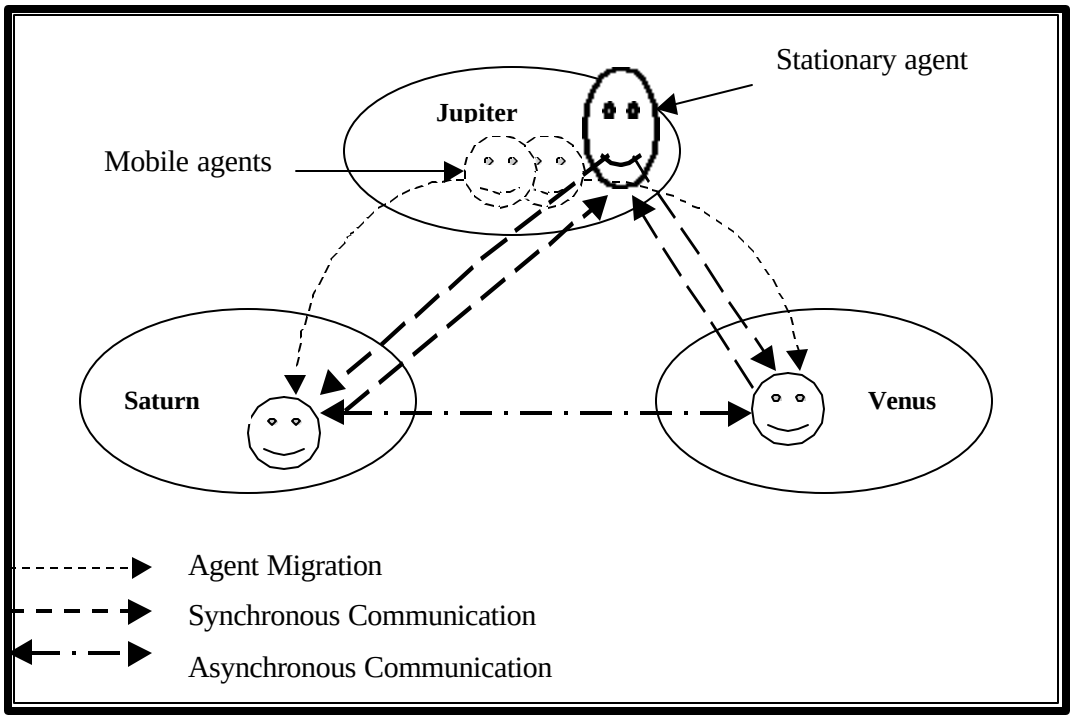

Figure 1: The Experimental setup of the application

2.2 The Agent Development Environment

As the agents are developed using the Aglet Software Development Kit (ASDK), this section presents a brief description of the development environment. The ASDK is a Java based mobile agent development toolkit. It supports creation, cloning, dispatching, retracting, and activation/deactivation of agents and facilitates the creation of dynamic itineraries (Yap et al., 2004). The main components of the ASDK are the Aglet API packages and a customizable agent server called Tahiti. The choice to develop the application using ASDK was based on its popularity, ease of use (Yap et al., 2004), (Gschwind et al., 1999) and it being freely available (at http://www.trl.ibm.com/aglets/download11b.htm).

The Aglet API is a set of Java classes and interfaces that allows for the development of mobile agents. The API uses the Java delegation event model introduced in Java 1.1 (Tai et al., 1999). This model is based on the concept of an "Event Source" and "Event Listeners". Any object interested in receiving an event is called an Event

Listener and the object that generates these events is called an Event Source. When an action is performed on the event source object, the event produced is then propagated to all event listeners. This is achieved by invoking a method on the event listener object and passing it an event object. For example, clicking the dispatch button (figure 2) invokes the listener's `onArrival` method and passes it the "`MobilityEvent`" object. A detailed explanation of underlying mechanism when such an action is taken is presented in section 2.3.1 below.

As mentioned above, the other main component of the ASDK is the Tahiti server, which supports the runtime execution of aglets. It provides the necessary infrastructure to support, amongst others, the creation and termination of agents, communication and security facilities. Figure 2 below presents the GUI of the Tahiti Aglet Server on Jupiter after creating and dispatching one of the mobile agents to Saturn.

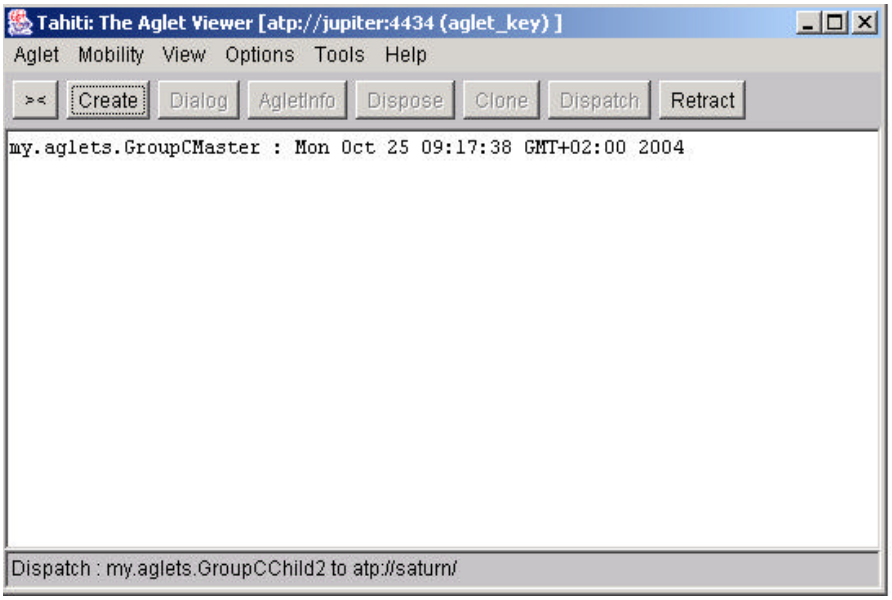

Figure 2: Tahiti aglet server.

2.3 Technical Details of the application

This section makes use of code segments taken from Appendix A to discuss the agent application. Extending the abstract class `Aglet` and overriding its `onCreation()` method creates the stationary agent represented by the Class `GroupMaster,` and

shown by figure 3. The code segment presented below illustrates this. Behind the scene, creation of an agent involves several steps. First of all, the class definition is loaded and the agent object is instantiated. The agent is assigned a unique identification by the *place.* In the next step, the agent is given a chance to initialize itself using any initialization arguments provided to it. Once the initialization has been completed successful can the agent start is execution independently of any other agent at that place.

```
public void onCreation(Object obj){
      window = new GroupWindow(this);
      window.show();
      try{
            name= getProperty("user.name");
         }
      catch(Exception e){
             e.printStackTrace();
       }
}
```

The GUI of the stationary agent consists of a frame with a text field and a text area. The message to be passed to other agents is entered in the text field and the messages received from other agents are displayed in the text area. The message handling mechanism is presented in section 2.3.2. In addition to these two main components, the GUI also consists of an address field that displays the address of the agent with which the stationary agent wishes to communicate. Figure 3 shows the GUI of the stationary agent produced by the class `GroupWindow`.

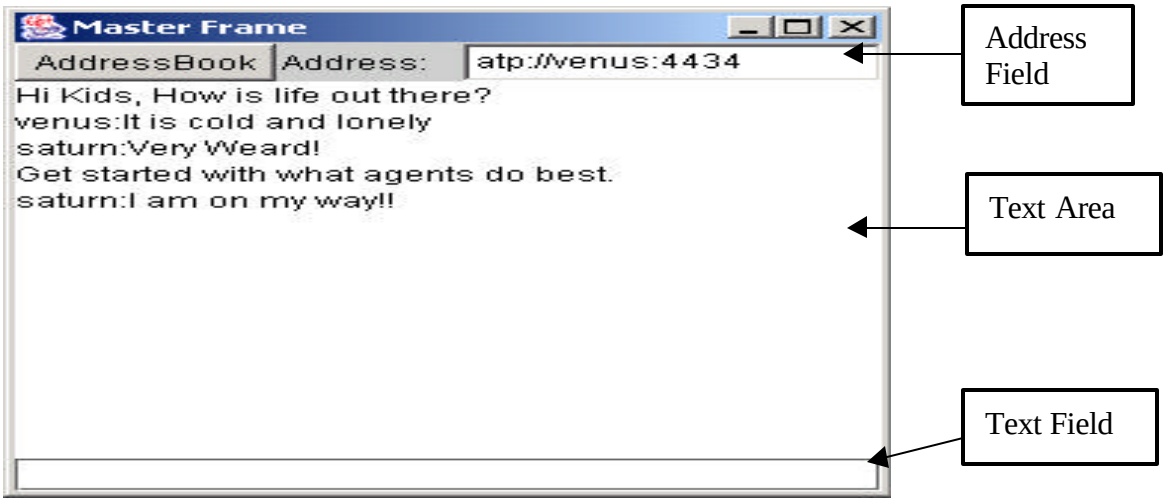

Figure 3: GUI of the stationary agent.

2.3.1 Agent Creation and Agent Mobility

The application requires that the stationary agent create two mobile agents and dispatch one to Saturn and the other to Venus. The two mobile agents that are created are called `GroupChild1` and `GroupChild2`. As required by Aglet and some other mobile agent systems, an agent is created at a "place" and moves between places. The concept of places is discussed further in section 4.4 of chapter 4. Places are known as "Context" in the Aglet system. Hence before a mobile agent is created, one has to gain access to the current context. The aglet context is also used to provide information about its environment, and to send messages to the environment and to other active agents in that environment. To send an agent to a particular host, for example to Saturn, the `dispatch()` method provided by the Aglet API is used. Basically, when an agent is dispatched to a particular host, its state information is preserved and a sequential byte representation of the agent is made by a technique known as object serialization (Sutherland, 1997). These bytes are then passed on to the underlying transfer layer, which carry it over the network. Using Java's reflection and deserialization techniques on the transferred bytes, the agent is reconstructed on the destination host (Karjoth et al., 1997). The above discussion is shown in the code segment below.

```
AgletContext ac1= getAgletContext();
AgletProxy ap1 = ac.createAglet(null,
"my.aglets.GroupCChild1",getProxy());
AgletContext ac2= getAgletContext();
AgletProxy ap2 = ac2.createAglet(null,
"my.aglets.GroupCChild2",getProxy());
remoteProxy1 = ap1.dispatch(url);
remoteProxy2 = ap2.dispatch(new
URL("atp://saturn:4434/"));
```

In accordance with aglet delegation-based event model, upon successful invocation of the `dispatch()` method, the listener's `onArrival()` method is called which is used to initialize the agent once it has arrived at the new destination. On arrival at a new host, the first thing the mobile agent does is to display its GUI (figure 4). The code segment shown below represents this discussion.

```
addMobilityListener(new MobilityAdapter(){
   public void onArrival(MobilityEvent me){
      try{
         window = new GroupWindow(GroupCChild1.this);
         window.show();
         name=getProperty("user.name");
      }
}
```

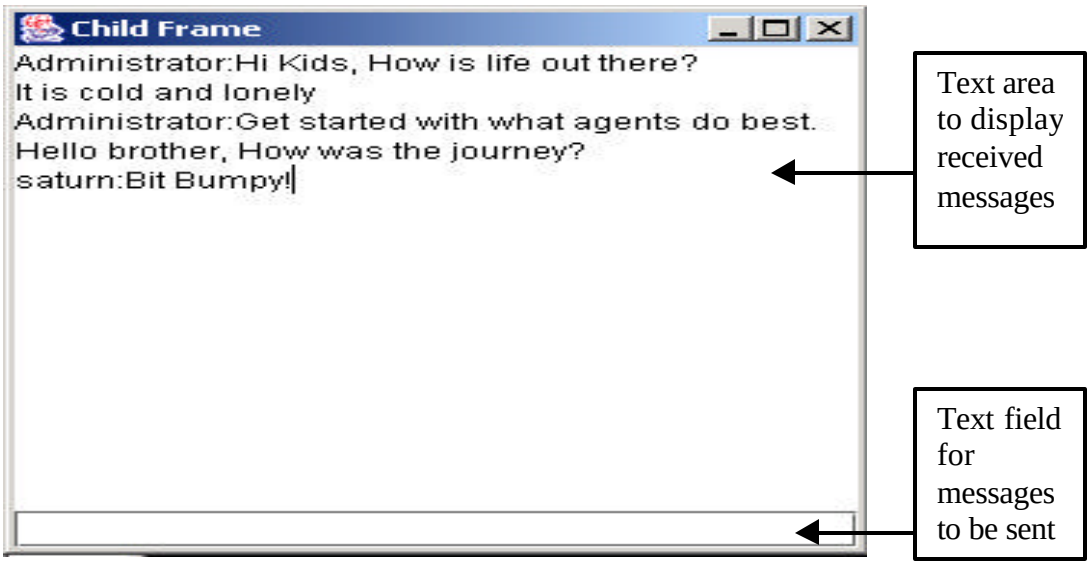

Figure 4: GUI of a mobile agent

In addition to being able to create and dispatch agents, agents need to get access to each other for various reasons, such as, for communication. It is a requirement of the Aglet system that all access to an agent takes place through a proxy. Proxy is a representation of an agent. It serves as a shied that protects the agent from malicious agents and also provides location transparency for the agent. Creating an agent is one way to get a proxy as shown by the code segment below.

```
AgletProxy ap=ac.createAglet(null,
"my.aglets.GroupCChild1",getProxy());
```

2.3.2 Communication between agents

Having the mobile agents dispatched successfully to Venus and Saturn, the agents begin to communicate. A requirement of this application is that the stationary agent communicates with the mobile agents by sending and receiving messages in a synchronous manner whereas the mobile agents communicate with each other in an asynchronous way. In synchronous messaging the sender of the message suspends its execution until a reply has been received whereas the sender continues its execution and retrieves the reply at a later time, in asynchronous communication. These requirements are placed on the agents purely for demonstrating the different types of communications that is possible between agents.

The principal way the agents communicate is by message passing. Inter-agent messaging is based on a simple event scheme that requires an agent to implement handlers only for the kind of messages that it is supposed to understand (Lange et al., 1998). For synchronous messaging, the `sendMessage()` method is invoked. This method takes a message object as a parameter, serializes the message using the object serialization technique explained in section 2.3.1 above and sends it to an agent it wishes to communicate with. The receiving agent then reconstructs the message using the deserialization technique. To respond to the message received, an agent makes use of the message handler `handleMessage()`. The code segments provided below demonstrate how a message is sent to the mobile agents and the mechanisms used by the receiving agents to handle the message.

```
if(remoteProxy != null) {
      remoteProxy.sendMessage(new
      Message("text",name+":"+s));
      remoteProxy2.sendMessage(new
      Message("text",name+":"+s));
}
```

```
public boolean handleMessage(Message message){
      if(message.sameKind("text")) {
            String s = (String)message.getArg();
            if(!window.isVisible())
              window.show();
            window.appendText(s);
            return true;
      }
```

Asynchronous messaging is based on the concept of future object. The object is called "future" because it is returned immediately to the sender of the message, bearing the promise of a future reply. The future object, which is returned by the `sendFutureMessage()` method, serves as a handle for the expected reply to the message (Lange et al., 1998). The code segment provided below shows how the mobile agents communicate in an asynchronous way.

```
future= ch1Proxy.sendFutureMessage(new
Message("text",name+":"+s));
   if(future.isAvailable()){
      String reply = (String) future.getReply();
      window.appendText(reply);
```

2.4 Summary

The chapter contains a description of an agent application used to demonstrate some of the means of achieving mobility and communication using ASDK and the underlying mechanisms to achieve them. A more comprehensive discussion with regards to mobility, communication and security are presented in subsequent chapters.

3 Agent Migration

3.1 Introduction

A *process* is a program in execution. The transfer of an executing program between machines is termed as *process migration*. From a low-level (operating systems) perspective, an *agent* is just a process, consisting of code and state. The main governing factor that distinguishes it from other processes is that in an agent not all of its instructions have to be executed on the same node or location. Just as a mobile agent is fundamentally different from a process (in this described sense), so too is agent migration different from process migration. The difference lies in *who* decides *where* to move. In process migration, migration is normally forced upon a process by the distributed operating system due to factors such as load balancing (Farooq et al., 2003), (Johansen et al., 1995a). A mobile agent's primary identifying characteristic is its ability to *autonomously* migrate from host to host (Picco, 2001), (Karnik et al., 1998). Thus providing support for agent mobility is a fundamental requirement of an agent infrastructure.

Agent mobility as is known today has evolved over time from simple data mobility as in the case of file transfer using FTP, to the transfer of execution control (in remote procedure call), to code transfer (as in remote invocation or code on demand), and currently to the transfer of execution and data state (as in mobile agents) (Zaslavsky, 2004), (Horvat et al., 2000). In section 3.2, a brief discussion of these paradigms is presented.

The agent migration process consists of deactivating the agent, capturing of its states, transporting the agent to the new place. Once the agent has reached its new host, the agent's states must be restored and the agent reactivated. Depending on the content of agent's state that is transported, different levels of mobility is obtained. This has been

discussed in section 3.3. Section 3.4 analyses the ways in which the codes needed by the agents are relocated.

On arriving at a new host, the set of bindings to resources available to the agent must be rearranged. This may involve removing bindings to resources, re-establishing new bindings or even migrating some resources to the destination node along with the agent. The option depends on the type of resources and on the requirements imposed by the application (Fuggetta et al., 1998). These issues are discussed in section 3.5.

Related to the issues of how an agent migrates, is the issue of where to an agent migrates and the transportation protocols used to support this. This set of sites to be visited, often called itinerary, can be either statically defined at the time of agent creation or dynamically built from the information the agent gathers as it migrates from node-to-node. The concept of agent migrating between hosts implies that the user is not reliant on the system that launches the agent and is not affected by the host failure. This implies that persistence is supported by the notion of mobile agents (Zaslavsky, 2004), (Harrison et al., 1995). However, specific mechanisms have to be embedded into mobile agent systems to handle situations such as breakdown of hosts, destruction of agents, or network errors leading the agents off course (Schoeman et al., 2003). These issues are discussed in sections 3.6, 3.7 and 3.8. Section 3.9 presents the factors to be considered when selecting mobile agent systems. These factors are based on issues covered in sections 3.1 to 3.8 and thereafter a summary of the chapter and conclusions drawn from it are presented in section 3.10.

3.2 Mobility Concept

Mobile agent systems typically identify the agent with a unit of execution running in an execution environment belonging to the lower level of the virtual machine, e.g. a thread or a process. A *unit of execution* (EU) is constituted by the *code segment* (CS) governing its behaviour, the *data state* necessary for its computation, and its *execution state*, such as program counter and call stack (Picco, 2001), (Fuggatta et al., 1998), (Gray et al., 2002). For instance, using the stationary agent `GroupCMaster` of

appendix A, the variables shown below form the data state and the rest of the code form the code segment.

```
AgletProxy remoteProxy;
AgletProxy remoteProxy2;
String name;
GroupWindow window;
AgletProxy chProxy;
Message msg3;
```

In a traditional distributed system, each EU is bound to a single Computational Environment (CE) i.e. the runtime system for its entire lifetime. Moreover, the binding between the EU and its code segment is generally static. Even in the environment that support dynamic linking, the code linked belongs to the local CE. In Mobile Code Systems (MCSs), on which most of the mobile agent systems are based (Tripathi et al., 2002), the code segment, the execution state, and the data space of a EU can be relocated to a different CE. Depending on the constituents of the execution unit that is relocated, two forms of mobility are identified in the literatures- strong mobility and weak mobility. Before discussing these two forms of mobility (discussed in sections 3.3.1 and 3.3.2), mobile code paradigms that provide different strategies for achieving distributed computing are examined. These paradigms consist of the following mechanisms:

3.2.1 Code on Demand (COD):

(Refer to figure 5 below) In COD, component C_A on host H_A has access to resource R. However C_A has no idea of the logic (code) required to perform the task. Thus C_A requests from component C_B on host H_B for the logic. Once the logic is received, C_A is then able to perform the task. An example of this mechanism is the Java applet.

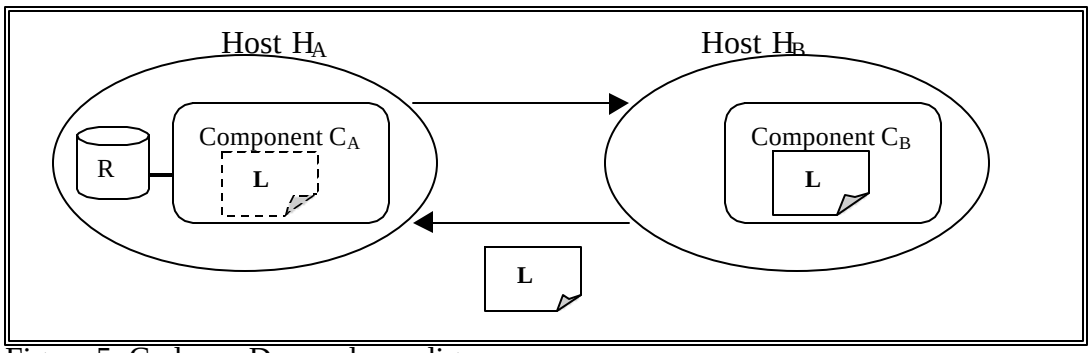

Figure 5: Code-on-Demand paradigm

3.2.2 Remote Evaluation (RE):

(See figure 6 below) In RE, component C_A on host H_A has the necessary logic (code) to perform a particular task, but does not have the required resources. The necessary resource R is located on host H_B, so C_A forwards the logic to component C_B on H_B where it is executed and the result is returned to C_A.

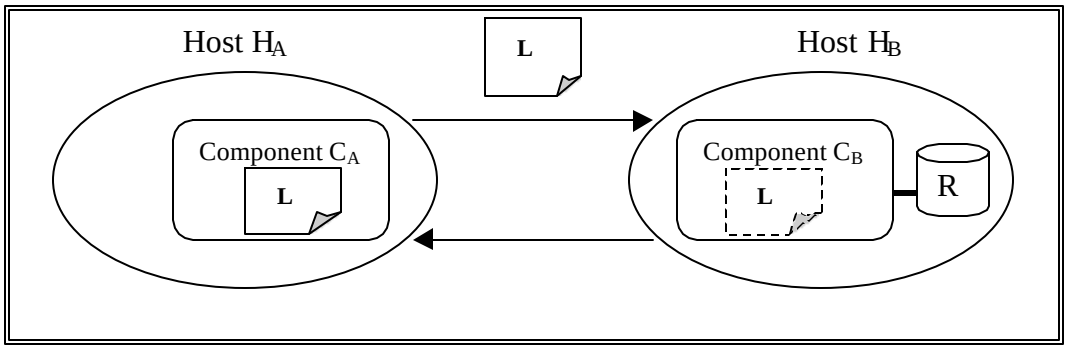

Figure 6: Remote Evaluation paradigm

3.2.3 Mobile Agent (MA):

(Figure 7) In this paradigm, component C_A has the necessary logic to perform the task, but lacks the resource R, which is at host H_B. Instead of passing the logic around, the entire component C_A is migrates with its associated data and logic to the new host H_B and interacts locally to access the resource.

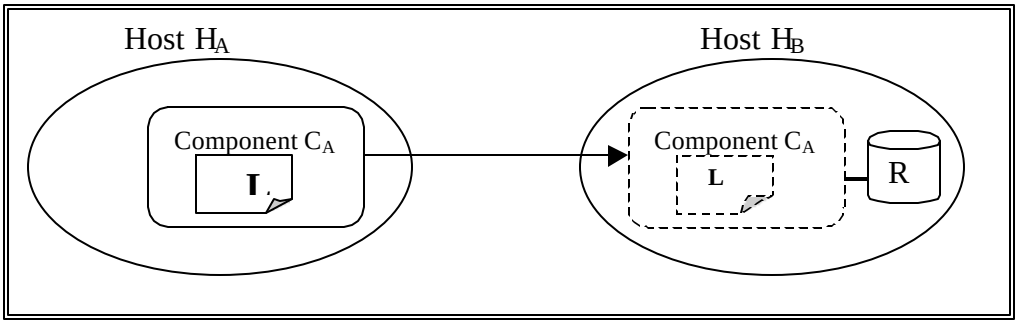

Figure 7: Mobile Agent paradigm

3.3 Mobility Models

Figure 8 below classifies mobility based on the components of mobile agents that is relocated when an agent migrates.

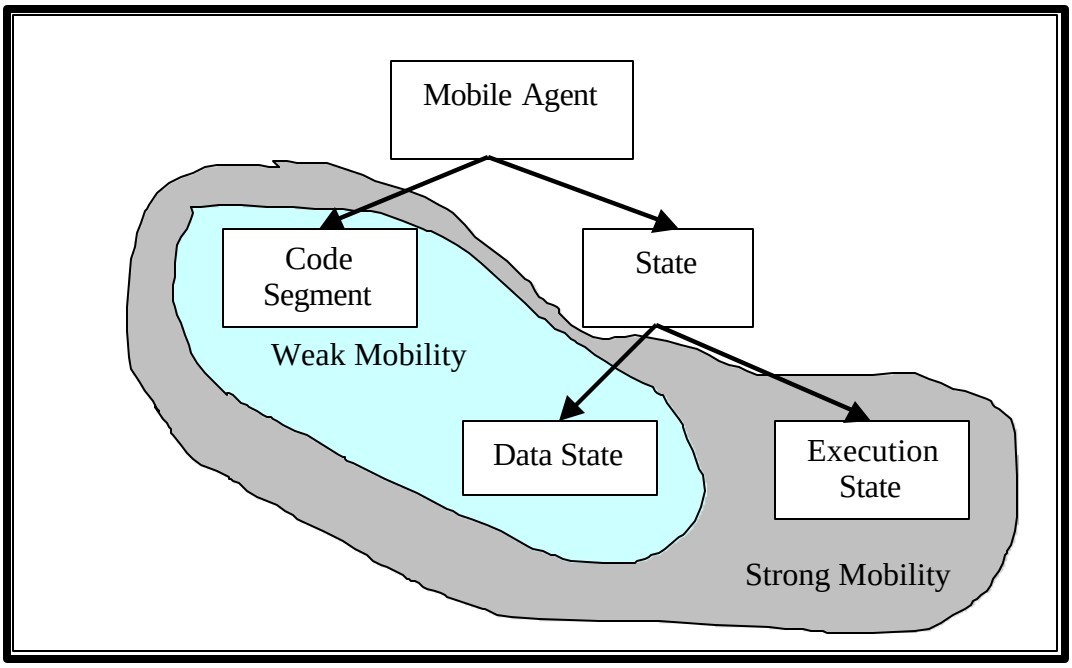

Figure 8: Classification of MAS based on agent component mobility

3.3.1 Strong Mobility

In strong mobility, all the three components - code segment, data state, and execution state (see figure 8 above) are captured and transferred to the destination machine (Brazier et al., 2002). Having the code to migration means that the code necessary to build up the agent is available at the destination site. Providing for execution state migration ensures the execution of the agent to begin at the same point where it had stopped before migration. Supporting data state migration (values of the variables placed on the stacks) ensures that the results of an agent's computation are available to the agent after migration.

As stated by Fuggatta et al., (1998) and Straβer et al. (1996), strong mobility is supported by two mechanisms: migration and remote cloning. In migration mechanism, once the command to migrate is issued, the execution of the agent at the current location is stopped; the agent is removed from the source location and transported to the destination location where the execution continues. In remote cloning mechanism, a copy of an agent is created at the destination host. Both these mechanisms could be either proactive or reactive. By proactive migration, it is meant that the migrating agent autonomously determines the time and the destination for migration, and reactive migration means that a different agent triggers the migration.

According to Arjav et al.(2003), even though such features are very powerful and interesting from a programmer's point of view, few systems implement a complete strong mobility. One of the reasons for this lack of implementation is due to the complexity in representing the states of an agent in a manner that is independent of the interpreter/compiler or heterogeneous platform on which an agent operates. Another difficulty associated with strong mobility is that if the code is compiled before migration, the execution state will need to be transformed into an implementation independent format to cater for different architectures, this may be a nontrivial task even if the same compiler is used. To overcome the problems associated with strong mobility, weak mobility is used.

3.3.2 Weak Mobility

Weak mobility is the ability of MAS to allow code transfer across different computational environments (CEs); code may be accompanied by some initialisation data, but no migration of execution state is involved. Mechanisms supporting weak mobility provide the capability to transfer code across CEs and either link it dynamically to a running execution unit (EU) or use it as the code segment for a new EU. As no execution state of an agent is captured in this mechanism, the agent after migration must restart its execution from the beginning or from a predetermined point - usually at a method (Thorn, 1997) and (Tripathi et al., 2002). For example, the mobile agent `GroupCChild2` (chapter 2) after migration, begins its execution from

the method `onArrival(MobilityEvent me)`. Some mobile agent systems support more than one point from where an agent can begin its execution upon migration.

According to Fuggetta et al.(1998), weak mobility can be classifies according to:

a) Direction of code transferred – either the code can be pushed or pulled to another CE.

b) Stand-alone code or code fragment – stand-alone code is self contained and can be used to instantiate a new EU. Code fragment needs to be linked to a running code to be executable.

c) Synchronization involved – depending on whether the EU requesting code transfer is suspended or not until the code is executed, mobility can be either synchronous or asynchronous.

d) Time of execution of the transferred code – when asynchronous transfer is used, the execution of the code transferred may take place either immediately or in a deferred fashion.

In systems using weak form of mobility, if the application requires the ability to retain the thread of control, extra programming is required in order to capture the execution state manually. As mentioned earlier and supported by the code segment of figure 9 below, in systems supporting strong migration, migration is completely transparent to the migrated program, which resumes execution right after the migration instruction. This has a double advantage of reducing programming efforts of using migration to the invocation of a single operation, and of requiring a smaller code size of the migrated code (Picco, 2001).

```
void main (String () args) {
……// some instructions

go( " NewHost", NewMethod");
 ……// not reached
} // end of main
void NewMethod() {
 // execution restarts here
 // on migration
……
}  // end of NewMethod
```

(a)

```
void main (String () args ) {
 ……// some instructions

    go("NewHost");
  // the execution restarts
  //  here on migration
    ……
  } end of main Method
```

(b)

Figure 9: Code fragment of an agent using (a) weak and (b) strong mobility

3.4 Strategies for Relocating Code Segment

The availability of the code (in Java – normally class files) of an agent at a host is essential for the execution of the agent at that host (Picco, 2001). There are various approaches adapted by mobile agent systems to the transportation of their classes to the host on which the agent is to execute (Gray et al., 2002), (Tripathi et al., 2001). One approach is that all classes required by the agent are transported with the agent when it is transferred. This prevents any further remote communication with other hosts while an agent is executing at a particular host. However, this makes the agent transportation heavyweight, since an agent may consist of large number of classes, and not all are actually needed for the task at hand on a particular host. The second approach is to have the classes pre-loaded at the various hosts. This approach is obviously impractical, as it requires a pre-knowledge of all the hosts in the system where an agent could migrate. The third scheme often adopted by some mobile agent systems is to migrate only the base class with the agent; additional classes are downloaded dynamically on need basis from a designated host. This avoids unnecessary transfer of classes, but imposes runtime overhead on agent's execution and it is not suitable in a disconnected environment (Tripathi et al., 2002). Another drawback of this scheme is its reliance on the assumption that the code repository is

always available, thus implicitly negating one of the major advantages of mobile agents, i.e. the ability to support disconnected operations (Picco, 2001).

Upon relocation of a EU to a new CE, its data space, i.e. the set of bindings to resources accessible by the EU, must be rearranged. This may involve removing the current binding to resources, re-establishing new bindings or even migrating some resources to the destination CE along with the EU. The choice depends on the nature of the resources, the type of binding to such resources, as well as on the requirements posed by the application (Fuggetta et al., 1998). In section 3.5 below, data management mechanisms used by various mobile agent systems are analysed.

3.5 Data space management

According to Fuggetta et al.(1998), the data space management mechanisms that can be exploited upon migration depends on the type of bindings involved between the resource and the EU and the nature of resources. Fuggetta et al.(1998) classify resources into three categories- *Free transferrable* (resource can be migrated over the network), *Fixed transferrable* (resources could be migrated but is not desirable, such as a huge or crucial file) and *fixed not transferrable* (Resources cannot be migrated over the network).

As suggested by Fuggetta et al.(1998), there are three ways in which resources can be bound to an EU:
The strongest form of binding is **by identifier**. In this case, the EU requires that, at any moment, it must be bound to a given uniquely identified resource. Binding by identifier is exploited when a EU requires to be bound to a resource that cannot be substituted by some other equivalent resource.

A binding established **by value** declares that, at any moment, the resource must be compliant with a given type and its value cannot change as a consequence of migration. This kind of binding is usually exploited when an EU is interested in the

contents of a resource and wants to be able to access them locally. In this case, the identity of the resource is not relevant; rather, the migrated resource must have the same type and value as the one present on the source CE.

Binding **by type** is the weakest form of binding. In this case, the EU requires that, at any moment, the bound resource is compliant with a given type, no matter what its actual value or identity are. This type of binding is exploited typically to bind resources that are available on every CE.

From the above discussions, as suggested by Fuggetta et al.(1998), two classes of problems must be addressed by data space management mechanisms upon migration of an EU: resource relocation and binding reconfiguration. The manner in which the existing mechanisms tackle these problems is constrained both by the nature of the resources involved and the form of binding to such resources.

Consider a migrating executing unit U whose data space contains a binding B to a resource R. One general mechanism, which is independent of the type of binding or resource, is binding **removal**. In this case, when U migrates, B is simply discarded. If access to bound resources must be preserved, different mechanisms must be looked at.

According to Fuggetta et al.(1998), when U is bound to R by **identifier**, two data space management mechanisms are suitable that can maintain resource identity. The first mechanism is relocation **by move** where R is transferred together with U to the destination CE and the binding is unchanged. Clearly, to exploit this mechanism R must be a free transferable resource. Otherwise, a **network reference** mechanism must be used. In this scenario, R is not transferred and once U reaches its target CE, modification is made to B to reference R in the source CE. The creation of inter-CE binding is often not desirable because it exposes U to network problems and makes it difficult to manage state consistency since data is actually distributed over the network. On the other hand, moving a resource away from its CE may cause problems to other EUs that own binding to the moved resource.

If B is **by value** and R is **transferable**, a suitable mechanism is data space management **by copy** because the identity of the resource is not relevant. In this case, a copy R of R is created, the binding of R is modified to refer to R , and then R is transferred to the destination CE along with U. Though binding **by move** also satisfies the requirement posed by binding **by value**, it may be less convenient as other ECs may be bound to the transferred resources. If R cannot be transferred, network reference mechanism is the only viable solution, with the drawbacks described above.

If U is bound to R **by type**, a suitable mechanism to use is **re-binding**. Re-binding makes use of the fact that the only requirement posed by the binding is the type of the resource must be available at the destination, and avoids resource transfers or the creation of inter-CE bindings In which case B is voided and re-established after migration of U to another resource R on the destination CE having the same type of R. (Fuggetta et al., 1998).

In summary, the relationships between the resources relocation, binding reconfiguration and the associated data space management mechanisms are as shown in the table below.

	Free Transferable	***Fixed Transferable***	***Fixed Not Transferable***
By Identifier	By Move (Network reference)	Network reference	Network reference
By Value	By copy (By move, Network reference)	By Copy (network reference)	(Network reference)
By Type	Re-binding (Network reference, By copy, By move)	Re-binding (Network reference, By copy)	Re-binding (Network reference)

Table 1: Binding, Resources and Data Space Management Mechanism (Fuggetta et. al, 1998).

3.6 Agent Transportation Mechanisms

While an agent is executing, it might determine that it needs to visit another site on the network. The most common way to achieve this is by invoking a migration primitive. The destination of migration specified by the agent could either be absolute or relative. In absolute migration the name of the host to which migration is to take place is specified, whereas in relative migration the name of another agent or resource is used. Referring back to the example of chapter 2, both mobile agents use the `dispatch` primitive and absolute migration destination to transport itself to a new host. The code segment below indicates this.

```
ap2.dispatch(new URL("atp://saturn:4434/"));
ap1.dispatch(new URL("atp://venus:4434/"));
```

Irrespective of the manner in which the destination of migration may be defined to transport the agent, a transport mechanism (protocol) is required. There are a number of protocols available to achieve this function.

Most of the Java based mobile agent systems make use of programming language features extensively, such as support for remote method invocation (RMI), object serialization, and reflection, which is implemented on top of TCP/IP or UDP/IP (Tripathi et al., 2001). One of the major drawbacks of this approach is its dependency on TCP/IP or UDP/IP and also the flat nature of the data that is passed around. On top of this, the agent must know about the protocol in use, which makes modifications to it difficult without redeployment of new agents (Papaioannou, 1999).

Some Mobile agent systems use HTTP protocol as a means to transfer agents. This is achieved by encapsulating agents as MIME contents (Satoh, 2001), (Lingnau et al., 1995). This content type could carry attributes to describe, for instance, the programming language used and the agent type. The agent server then uses this information for choosing the right kind of runtime support. The advantages of using the widely accepted Internet protocol are platform-independent mechanism, ease in the use of the agent-based services, and access to future advanced features of the next

generation HTTP such as security services. A few agent systems, for instance Voyager (Horvat et al., 2000), make use of CORBA's Internet Inter-ORB protocol (IIOP) and/or Microsoft's Distributed Common Object Model (DCOM) for agent transport. Yet other systems, such as Mole (Uhrmacher et al., 2000) use other protocols, like FTP and SMTP to get the agents across. In addition to the protocols mentioned above that are used to transport agents, some mobile agent systems implement their own agent transfer protocols at an application level to assist in transportation of the agents. For example, the mobile agent `GroupCChild1` uses Aglet's Agent Transfer Protocol to transport the agent to Saturn.

```
ap1.dispatch(new URL("atp://saturn:4434/"));
```

3.7 Agent Persistence Mechanisms

In a traditional client-server environment, the state of the computation is distributed between the client and the server. This kind of system could experience problems due to failure of either the client or the server components. As mobile agents have their states centralised within themselves, an improved fault-tolerance can be expected from this technology. When moving through a large and unreliable network such as the Internet, mobile agents may fall victim to multiple accidents such as host crashing or line breakdowns. Explicit persistence mechanisms should be incorporated into the mobile agent systems to avoid loops and endless waiting for agents to return to their home base. However, fault tolerance is often neglected by most mobile agent systems (Picco 2001), (Straβer et al., 1998), and (Assis et al., 1998).

Generally, fault-tolerant systems are constructed by providing recovery mechanisms at two levels. One of them is to perform system level error handling and recovery in a manner which is largely transparent to the application. The mechanisms at this level could be based either on creating checkpoint or on replication (Gray et al., 2002), (Picco 2001), (Tripathi et al., 2001), and (Johansen et al., 1999). In checkpoint

mechanism, the agents state information is checked (i.e. a complete record of its current internal state, at any time in its execution, are stored on some persistence media) before and after execution on a host and when the host recovers after a shutdown. The agent's states are restored from the saved record. Replication makes use of replication servers to mask failures (Pleisch et al., 2003).

The second level is at the application level where error conditions in an application signals the execution of an application-specific exception handler. If an agent encounters an exception that it cannot handle, its server can take suitable actions to assist the application with recovery. For example, the server can send a notification to the agent's owner, which can recall the agent or terminate it. Alternatively, the server can simply transfer the agent back to the owner, which lets the owner inspect the agent's state locally and restart it with appropriate corrected state (Tripathi et al., 2002).

3.8 Agent Itinerary

In a typical scenario, a mobile agent has to visit more than one host in a network to fulfil its task. An important question in this regard is how does an agent pick a host to move to? In the simplest case, a mobile agent can carry along a "travel plan", provided by its programmer/owner, consisting of the places to be visited one after another while attending to its chores. Referring back to the example of chapter 2, if the intention was to migrate an agent from Jupiter to Saturn via Venus and not sending separate agents to Venus and Saturn, addition of the following code segment would have achieved the objective.

```
Vector itinerary = new Vector();
itinerary.addElement(new URL("atp://venus:4434/");
itinerary.addElement(new URL("atp://saturn:4434/");
Enumeration enum = itinerary.elements();
while (enum.hasMoreElements()) {
   proxy= proxy.dispatch((URL)enum.nextElement());}
```

However, due to the size and the dynamical nature of network in which the agents have to operate, migration strategies cannot be derived form a-priori assumptions with respect to resources and nodes that are available at agent execution time and hence mobile agents must be developed with the ability to construct and proactively adapt their own travel plans (often called its itinerary) (Cabri et al., 2000a).

The itinerary is a data structure composed of multiple destinations, each destination describes a location to which an agent travels and the task the agent is to perform at that location. This could be static (fixed at the time of mobile agent initialisation – as shown in the code segment above), or dynamic (determined by the mobile agent logic) (Cabri et al., 2000a). To facilitate dynamic adaptability of travel plans, policy-based approach is adapted by many of the mobile agent systems. Policies are rules governing choices in the behaviour of a system separated from the components in charge of their interpretation (Yen et al., 2004), (Wies, 1995). In the context of mobility, policies can be specialised to specify choices in the mobility behaviour of agents in terms of when and where an agent has to migrate. Using policy-based approach has an advantage that it provides a clear separation between migration rules and agent codes, thus permitting a change in agent mobility behaviour without re-implementing agents themselves. The ability to dynamically modify their itinerary at runtime, provides flexibility whereas static itinerary facilitates ease of implementation.

Some of the mobile agent systems allow for multiple entry points in their itinerary models while others restrict to a single entry point only. According to Wong et al.(1997), single entry point model unnecessarily presents the agent programmer with a different programming model than that of the non-mobile programming paradigm. For complex agent applications, this constraint can require the programmer to maintain a large amount of state information that can be better encapsulated within the agent's itinerary.

Section 3.9 presents the factors, based on the discussions presented in preceding selections, used in selecting mobile agent systems. Chapter 6 uses these factors to analyse a few mobile agent systems.

3.9 Evaluation Factors

1) Mobility Model supported (Strong / Weak):

2) In systems supporting strong mobility-

   a) Manner in which agent's mobility is achieved (migration / remote cloning):

      This factor determines whether the MAS migrate the whole agent or create a clone of the agent at the remote host.

   b) Nature of migration / remote cloning (proactive / reactive):

      This factor helps in evaluating the migration mechanisms supported by the agent systems.

3) In systems supporting weak mobility-

   a) Direction of code movement (push / pull):

      This factor determines whether the code is shipped or fetched from a remote site.

   b) Nature of code migrated (stand-alone code / code fragment):

   c) Synchronisation involved (synchronous / asynchronous):

   d) Time of execution (immediate / deferred):

      This factor determines whether the execution of the agent (in the case of asynchronous transfer) is immediate or delayed.

   e) Single or multiple entry points (single / multiple):

      After an agent migrates to a host, this factor determines whether the system supports a single or multiple methods as a starting point of execution.

4) Strategies used for code relocation (heavyweight / lightweight / preloaded):

   This factor determines the strategies for code relocation to determine whether the mobile agent system carries with is the entire code (heavyweight), only the

base classes (lightweight) when it migrates or are the codes preloaded on various hosts.

5) Data space management techniques (Binding removal / Network reference / Re-binding / By copy / By move):
This factor helps evaluates the kinds of data space management mechanisms supported by the MAS upon migration.

6) Nature of agent migration (absolute / relative):

7) Agent transportation mechanism (standard and / or proprietary):
This factor help explore the transport mechanisms for agents to determine whether a standard or proprietary mechanism is used.

8) Agent transportation protocols used (list of protocols):

9) Agent persistence-
   a) The level at which the MAS supports persistence (system and/or application):
   b) Mechanisms used for supporting agent persistence (list of mechanisms used):

10) Nature of the agent's itinerary (static / dynamic):
This factor examines the agent's itinerary to determine whether the MAS support itinerary statically defined or dynamically built.

11) Ability of one mobile agent to migrate another (yes/no)

3.10 Conclusions and Summary

From the discussions presented in this chapter, the advantages and disadvantages of strong and weak mobility is quite clear, however deciding on which of these mechanisms to choose from depends on the application to be developed. For example, if the application involves solving the problem of distributing the load amongst several distributed processes (for load balancing purposes), agent systems providing strong mobility is an obvious choice. In a load balancing application, it is required that the application be restored to the exact state before the movement of the agent and that it be transparent to the application itself. Another application where agents are considered to be useful is in information retrieval applications, where mobile agents visit a set of sites and search for given information. This class of application is an example of agents applying the same algorithm (in this case a search algorithm) on each site visited. In such cases mobile agent systems supporting weak mobility would be sufficient. This is because the agents need not be restored to the exact state before their migration. In majority of applications mobile agent systems supporting weak mobility is adequate. This is supported by the fact that most of the agent systems are developed in Java (see agent list at *http://mole.infomatic.uni-stuttgard.de/mal/preview.html)*. It is to be noted that the standard Java Virtual Machine supports only weak form of mobility (strong form of mobility is supported by modification of the JVM).

With respect to agent itinerary, it is recommended that the agent systems support both static and dynamic itinerary for reasons presented in section 3.8. The support for agent persistence in mobile agent systems is an important issue to be considered while deciding on a particular agent system. For agent transportation, it is recommended that the mobile agent system support several transportation protocols to make it applicable to real world scenario.

This chapter presents the various paradigms supporting distributed computing. A classification of agent mobility based on the components of the agent that is relocated together with the effect it has on various issues related to MAS has been presented. In addition, the chapter also presents the types of resources, bindings and the data space

management mechanisms. A study of the transportation mechanism (protocols) for the agents has been presented. Issues related to supporting fault-tolerance in agent systems, providing a “travel plan” for mobile agents has also been reported. In section 3.9, a list of factors to be considered when evaluating mobile agent systems are presented. The next chapter reports on issues surrounding agent communication.

4 Agent Communication

4.1 Introduction

A major setback for mobile agent technology apart from a frequently cited absence of appropriate security mechanism is the lack of interoperability between diverse mobile agent systems (Claessens et al., 2003), (Pinsdorf er al., 2002), and (Karnik et al., 2001). More precisely, two mobile agent systems are considered to be interoperable if a mobile agent of one system can migrate, interact and communicate with the agents on the second system (Pinsdorf et al., 2002). To accomplish useful tasks, agents often must communicate and collaborate with each other in an integrated fashion. According to Gray at al. (2002), there are four important issues regarding communication to be considered in the design of a mobile agent system. They are as follows:

- Identifying the party with which to communicate.
- Delivery of messages between the parties.
- Maintenance of communication and
- Required communication format

Inherent in mobile agent communication are the issues surrounding agent identification. A mobile agent, in most cases, has to be uniquely identifiable to achieve meaningful communication. To achieve this goal of meaningful communication, besides the agents being identifiable, the agents with which to communicate need to be located. The issues regarding agent identification and how they are found are discussed in sections 4.2 and 4.3.

After discussing the various means of identifying and locating agents, in section 4.4, a model for agent communication has been presented with an intent to study the types of agent communications. In addition, section 4.4 also presents the manner in which

the communications with these mobile agents are maintained. Here, communication issues that need support from the underlying Network layer are dealt with.

Section 4.5 analyzes the format of agent communication from a level above the ones discussed in the preceding sections, which is concerned with the transmission of bits and bytes. This issue must be taken into consideration especially if the agent has to operate in a heterogeneous environment. According to Rovatsos et al.(2004) and Finin et al. (1998), an agent communication language (ACL) provides the capabilities to integrate disparate source of information originating due to differences in the agent's operational environment. Section 4.6 presents the factors to be considered when selecting mobile agent systems and thereafter section 4.7 presents the conclusions and summary.

4.2 Identification of Agents

To assist in control, communication, coordination and cooperation of agents to take place, agents must be identified uniquely in the environment in which they operate.
To achieve this, the mobile agent systems need to come up with some means to assign globally unique identifications to their agents. This is not a new requirement as far as distributed system paradigm is concerned. In the client-server paradigm, for example, both Microsoft DCOM components and CORBA objects use unique identifications. This is usually a sequence of numbers and letters 128 bits long generated by DCOM or CORBA utilities. The objects in DCOM or CORBA are not referenced directly but rather make use of proxies or stubs (in the case of CORBA). This mechanism cannot be used in Mobile agent system, as the agents may need to be directly referenced for various reasons. For example, an agent may need to be called back by its architect to its home platform. Remembering a string of unique numbers to do so is not straightforward for human programmers.

All mobile agent systems make use of some identification scheme that generate unique identifiers and establish some infrastructure to allow convenient and accurate access to the agents that carry them (Tripathi et al., 2002), (Plam et al., 1998). Some agent systems make use of IP address and port number to assign a unique identification to their agents and use Domain Name Services (DNS) to resolve them. Others support federated naming service that facilitates linking of directory services to form a large logical directory (Domain Name System style). Yet other systems use simple table look-up to associate name string to URL.

The methods to identify the agents uniquely mentioned thus far introduce location-dependency. Though, using location dependent naming schemes have the advantage of being simple to implement, it makes the application's task of locating its agent more difficult because as the agent migrates, its name changes (Corradi et al., 2001), (Karnik et al., 1998).

To provide for location transparent naming at an application level, the mobile agent systems make use of either local proxies (as in Aglets and Voyager) for remote entries, which encapsulates their current locations, or use global location independent names (as in Ajanta) that do not change when the entity relocates.

When uses of local proxies are made, support by the runtime environment must be provided to update the entries in the proxy whenever an agent migrates. It must be noted that, doing this, creates a strong binding between application level names and network level names and requires a separate directory service to map various agents to URLs. Also, this raises the issue of performance if there are a large number of proxies in the network for an agent (Corradi et al., 2001).

Making use of global location independent naming schemes simplifies the programmer's task significantly, because a program can be written without regard to the current location of a mobile entity. It is important that the naming service operates robustly and securely to prevent tampering with agent's execution or the name service database (Tripathi et al., 2002).

Being able to identify the agents uniquely is not enough for communication to take place; one has to be able to locate the agent with which to communicate. The next section deals with this aspect.

4.3 Locating Agents

As mentioned in the previous chapter, an agent could be on any of the numerous nodes in a network due to its ability to migrate autonomously. If an agent wishes to communicate with another agent, it will need to locate the agent. For example, when an agent wishes to request a service from another agent. Therefore, it is important that agent systems must provide some mechanisms to locate an agent. There are various ways in which Mobile agent systems go about achieving this.

Some agent systems make use of preordained migration paths along which an agent migrates (Baumann 1999), (Chang et al., 1997). Probing at different points along this path, an agent can be located. This can be done either sequentially (probing one place at a time) or in parallel (probing more than one place at a time). On the other hand, when there is no preordained migration path to be followed, i.e. when the mobile agent is autonomous, then probing is not practical. In situations such as these, a variety of logging techniques are used in which information that ultimately leads to the place on which the agent reside are stored. This information can be stored directly in a database or in path proxies.

According to Baumann and Rothermel (1998) and Pitoura et al.(2001), when databases are used to store agent location information, the designers of agent systems have two choices. The database could be implemented globally or they could be implemented locally on each node. Using a global database means that the agent can be located by simply querying this database. An obvious disadvantage of this approach is that locating an agent depends on the availability of the database including the availability of the node on which this database resides and the availability of the communication channel to this node. To overcome the potential

bottleneck problem, the agent information can be stored on different local databases. This leads to a problem of identifying the right database from which the relevant information could be retrieved. A further problem is that the agents will need to negotiate with the local nodes to store their information. Some nodes may not be willing to allow its resources to be used in this way for security reasons. In some agent systems, the agents advertise their location whenever it deems necessary, this information is then used to update a database.

By storing a pointer to the target place of an agent's migration on every machine that it visits, a path of proxies is created. Following this path will eventually lead to the agent. Since the information is stored along the path, no additional communication is necessary to maintaining the path Fang et al.(2004), (Jul et al., 1988).

In those agent systems that do not provide for logging agents location information, brute force is used to search for the agents. This search can be done sequentially, one place at a time, or in parallel where a broadcast is made to every place in the system (Taha et al., 1999). The agent is located much faster when broadcasting is used but this leads to a much higher volume of messages being transmitted (Corradi et al., 2001), (Milojicic et al., 1998).

Aridor and Oshima (1998) suggest two methods for delivery of messages between entities interest in communicating. First one being the locate-and-transfer process whereby the agent is located using one of the mechanism discussed above and then the message is transferred directly to it. The second method being the forwarding mechanism whereby locating the receiving agent and delivering the message occurs in a single phase. The first process of messaging may have the disadvantage that the agent may migrate before the start of the second phase and the second has the disadvantage that it is less efficient for large messages (Schoeman and Cloete, 2003).

It must be noted that most of the mobile agent systems implement a combination of mechanisms discussed above for identifying and locating agents. In the next section, a minimalist communication model is presented with the intent of discussing the different types of communication in which agents could be involved.

4.4 Types of Agent Communication

To identify key components of MAS and to facilitate discussions on issues surrounding agent communication, use of an agent communication model has been made. For this dissertation, the model (figure 10) as presented by Baumann et al.(1997) has been used.

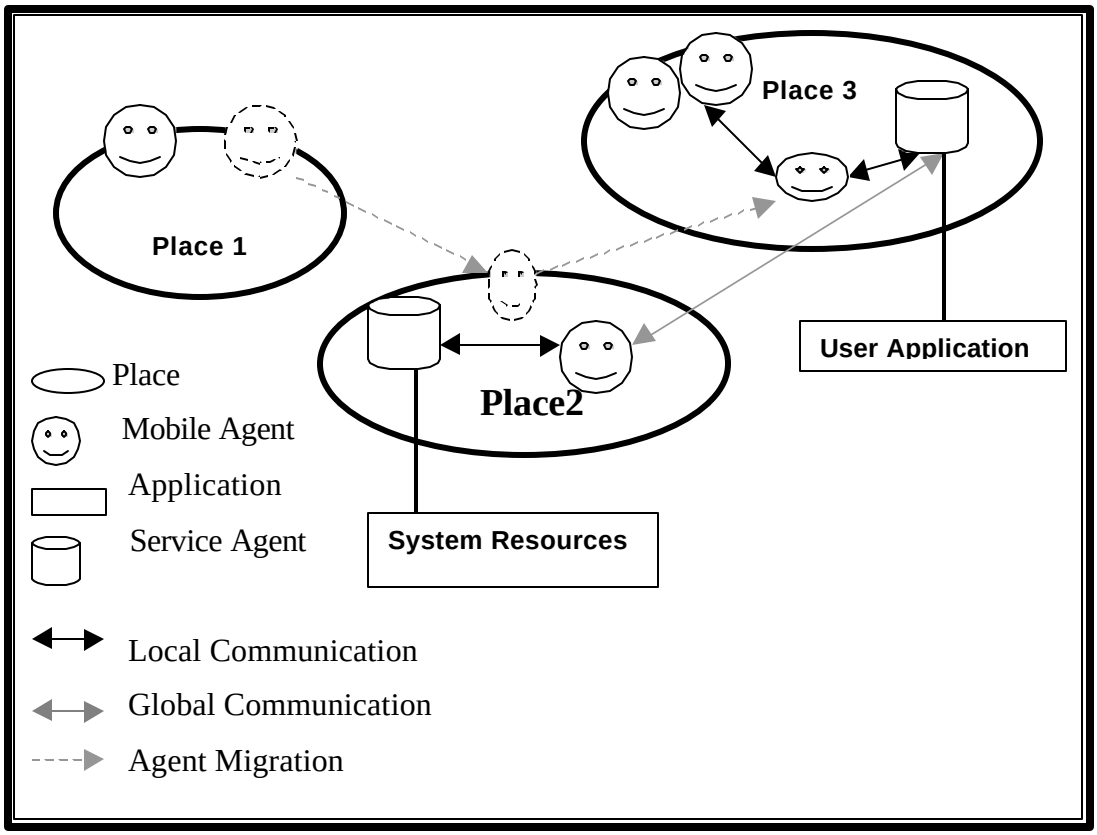

Figure 10: Agent Communication Model

Apart from agents themselves, one of the components of this model is the *place.* Places are the habitat of various services offered by the underlying systems (Fischmeister at al., 2001). Places are uniquely identifiable and serve as destinations for agent migration. Agents being active entities may move from place to place in order to meet other agents and to access services provided on those places. Places not only provide the services but also a safe execution environment for executing local as well as visiting agents. Security related issues are presented in the next chapter.

Another important component of this model is the so-called *service agents.* Unlike mobile agents, service agents are stationary and interface the services available at

places. Technically, service agents provide a mapping between the service request (expressed in "agent language") and the individual service interface in a manner similar to the ways in which DCOM or CORBA provide for heterogeneity in programming language representation of a request (Wolfgang, 2000).

Based on above discussions, agent communication can be broadly divided into the following categories as depicted by the figure 11 below.

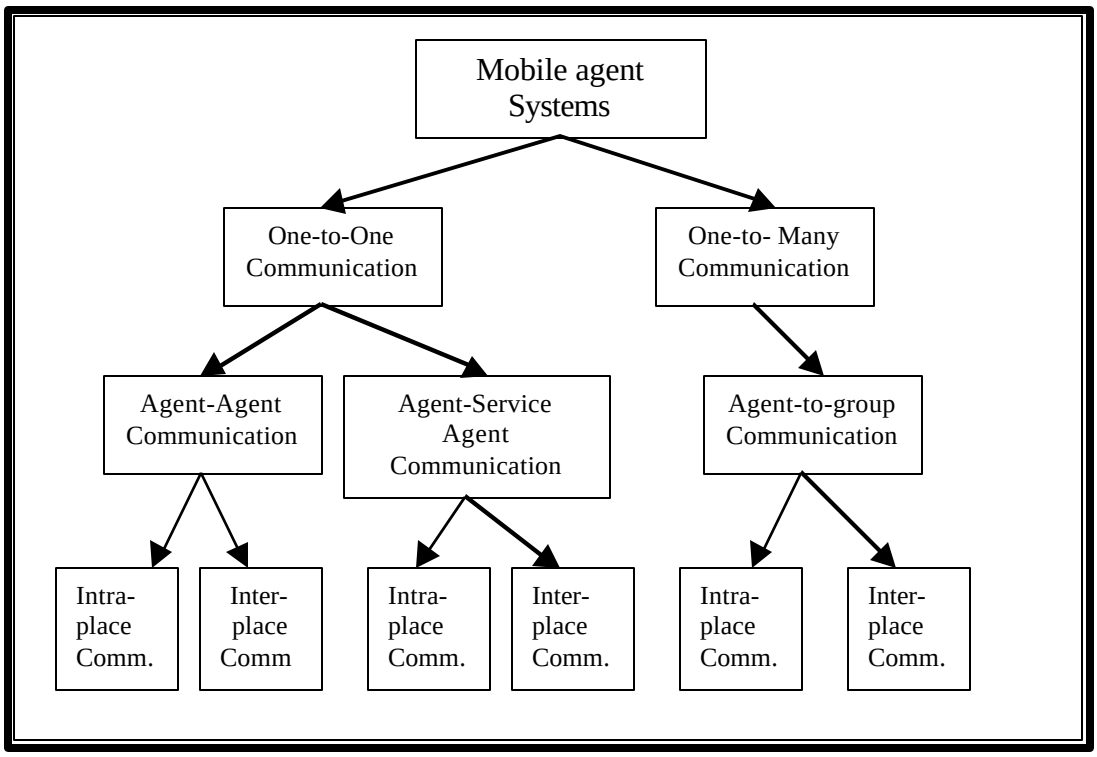

Figure 11: Types of Mobile Agent Communication.

4.4.1 Intra-place Communication (Mobile Agent – Mobile Agent)

Each mobile agent has its own plans, which it must initiate and control, in order to fulfil its needs and goals. Agents collocated at a place use this opportunity to provide references to each other that could be used to invoke the desired operations. The communication patterns that may occur in this type of interaction are peer-to-peer and are not limited to only request/response (Cabri et al., 2001), (Rothermel et al., 1998).

The communication mechanisms used by most mobile agent systems are based on RPC although some systems use message passing and a few use streams as well (Luc,

1999). Streams are a communication mechanism that allows a EU to open a channel to another EU, and send it a continuous stream of data.

4.4.2 Inter-place Communication (Mobile Agent – Mobile Agent)

For agents that are not located at the same place, a common approach to communication among agents takes place through the exchange of messages. The messages may be synchronous or asynchronous. With reference to the application of chapter 2, the stationery agent on host Jupiter communicates with each of the mobile agents on different hosts in a synchronous manner. This is achieved by using the primitive `sendMessage()` to send the message as shown in the code fragment below.

```
remoteProxy.sendMessage(new Message("text",name+":"+s));
remoteProxy2.sendMessage(new Message("text",name+":"+s));
```

The code fragment below shows how the agents receive the messages.

```
public boolean handleMessage(Message message){
   if(message.sameKind("text")) {
      String s = (String)message.getArg();
      if(!window.isVisible())
          window.show();
      window.appendText(s);
      AgletProxy ch1Proxy = AgletProxy)message.getArg();
      return true;
      }
}
```

Asynchronous communication between the mobile agents `GroupCChild1` and `GroupCChid2` is achieved by using the `sendFutureMessage()`as indicated below.

```
ch1Proxy.sendFutureMessage(new Message("text",name+":"+s));
```

4.4.3 Intra-place Communication (Mobile Agent – Service Agent)

Since service agents are representatives of services in the agent world, the style of interaction is typically client-server in which service agents provide operations or methods that can be requested by other agents. For this purpose an RPC-like communication mechanism is incorporated in the agent system. If the mobile agent system is developed and used exclusively in a Java environment, RMI is then used as the communication mechanism.

4.4.4 Inter-place Communication (Mobile Agent – Service Agent)

To locate or enquire about a particular service, mobile agents use this type of communication. Direct message passing is the predominant mechanism for this purpose Fang et al.(2004), (Peine et al., 1997).

4.4.5 Intra-place Communication (Agent-Group)

The previous four types of communication assumes that the communicating partners know each other when communication takes place i.e. the sender of a message or RPC is able to identify the recipient. However, there are situations, where a sender does not know the identities of the agents that are interested in the sent message. For example, a given task is to be performed by a group of agents, each agent taking over a subtask. In order to perform their subtasks, agents itself may dynamically create subgroups of agents. In order for an agent to communicate to the entire subgroup, MASs support several types of mechanism (discussed below).

Shared memory is often a popular choice in such situations. In order to communicate, several EUs are given a reference to the same variable or object. Changes in the value associated with the object are perceived by all the EUs owning a reference to it.

Some mobile agent systems use event-based mechanism for this purpose. In this mechanism, an event bus is defined which forms a logical channel through which events together with primitives that allow EUs to generate events and to subscribe for receiving events are dispatched. Upon generation of an event, the communication systems send a copy of the event description to all EUs that have subscribed to it. It must be highlighted that using this mechanism has the drawback of the messages getting lost in the presence of agent mobility (Cabri et al., 2001). Many variants of the event-based mechanism exist. One such variant combines the notion of events with a notion of groups. Groups provide a naming scheme that can be used to support multicast (Cugola et al., 1998).

Another form of implicit communication is provided by an abstraction of a communication channel called tuple space (Cabri et al., 2001). A tuple space is a shared dataspace with data organized as ordered sets of typed fields called tuples. EUs communicate by either inserting the tuples containing the information to be communicated into the shared space, or by searching it for a tuple using some form of pattern matching. This mechanism provides a very powerful abstraction of a communication channel (Cabri et al., 2000b). According to Tripathi et al. (2002), tuple space mechanism is not suited for bulk data exchange. A variant of the tuple space mechanism is the blackboard model. In this model, the agents interact through shared message repositories at each place, called blackboards, i.e. the sender puts a message on the blackboard and the receiver can either read or retrieve the message from the blackboard. In contrast to blackboard concept, tuple space provides additional access control mechanisms (Omicini et al., 2001), (Rothermel et al., 1998).

4.4.6 Inter-place Communication (Agent-Group)

Although there has been a lot of research conducted between agents and agent groups with respect to inter-place communication (Chang et al., 2003), (Yen et al., 2004), from the extensive literature survey conducted, it was not possible to conclude if there are any mechanisms implemented that provides support for this category of communication.

4.5 Agent Communication Languages

The previous sections dealt with communication between agents from a low level perspective where transportation of bits and bytes are considered. Agents need to ask other agents, to inform them, to request their services and find other agents to assist them in their tasks, and so on. Such functionality cannot be provided by simple mechanisms such as message passing or Remote Procedure Call. Agents need an Agent Communication Language (ACL) in order to achieve this. The main objective of an ACL is to model a suitable framework that allows heterogeneous agents to interact, to communicate with meaningful statements that convey information about their environment (Kone et al. 2000). Providing for agent communication at this level depends a great deal on the format of communication involved which in turn depends on the environment in which the agents operate. If an agent is meant to operate only in a homogeneous environment, then issues related to interoperability does not arise and hence mechanisms dealing with the ontology (vocabulary for agent communication and a set of relationship between the vocabulary items) (Fikes et al., 1999) are easy to implement.

Because mobile agents also operate in heterogeneous environment, interoperability cannot be achieved if there is not a standardized means through which agents can understand and communicate with each other and their environment. One such mechanism is as provided by CORBA/DCOM architecture that makes use of Interface Definition Language (IDL) to provide mapping for several languages. ACLs stand a level above CORBA/DCOM because they handle propositions, rules and actions instead of simple objects (with no semantics associated with them). ACLs messages describe a desired state in a declarative language, rather than a procedure or a method.

At a technical level, inter-agent communication can be seen to take place at the following levels: At the top level, content of messages exchange between the communicating parties is defined. The level below this is the message transport level, which contains various protocols that are used to transfer messages between the communicating peers. This level also provides for location transparency by mapping agents identity and physical locations. The lowest level contains the actual network transportation mechanisms.

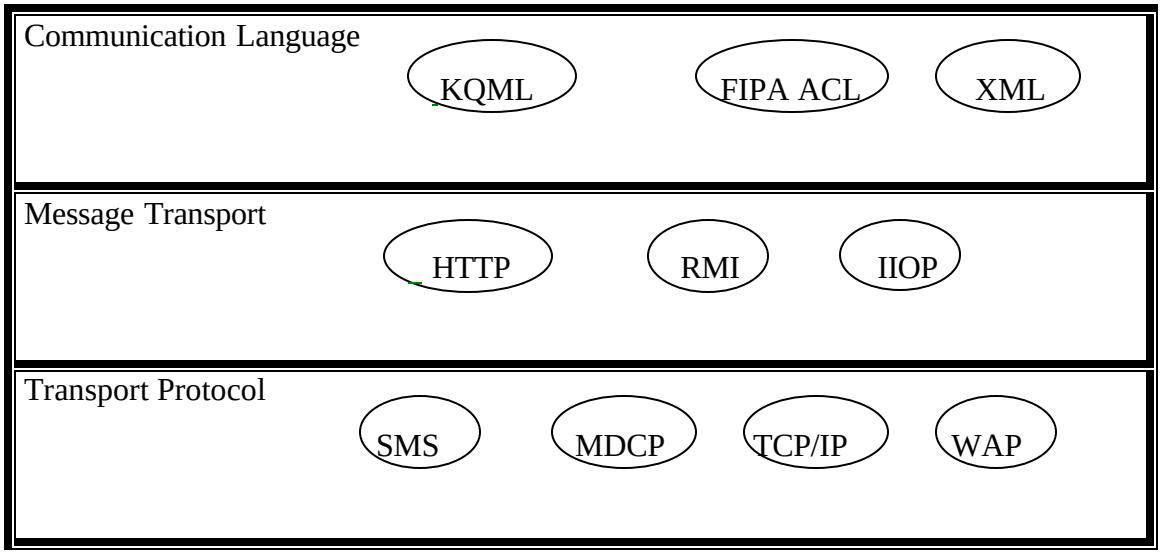

Figure 12: Agent Communication Layers (based on Helin et al. available at http://www.cs.helsinki.fi/)

Not all mobile agent systems are based on CORBA architecture, in those cases, alternate means to support communication amongst heterogeneous agents need to be used. Sections 4.5.1 to 4.5.3 present some of the ways to achieve communication in a heterogeneous environment.

4.5.1 Knowledge Query Manipulation Language (KQML)

One of the earlier results of research to provide a common agent communication language to describe and process agents collaboration request was KQML (Wooldridge, 2000). As stated by Labrou et al.(1999), KQML is a high level, message-oriented communication language and protocol for information exchange, independent of content syntax and application ontology. In other words, KQML is independent of the transport mechanism (TCP/IP, SMTP, IIOP, etc.), independent of the content language (KIF, SQL, STEP, prolog, etc.) and independent of the ontology assumed by the content.

4.5.2 Foundation of Intelligent Physical Agents - Agent Communication Language (FIPA - ACL)

According to Chaib-Draa (2002), a second attempt to support communication amongst heterogeneous agent produced the FIPA-ACL. The FIPA-ACL, like KQML, is based on speech act theory: messages are actions, or communication acts, as they are intended to perform some action by virtue of being sent. KQML and FIPA-ACL are almost identical with respect to their basic concepts and the principles they observe and differ primarily in the details of their semantic framework. The main distinguishing factors between ACLs like KQML and FIPA-ACL and past such efforts (RMI or CORBA) are the object of discourse and their semantic complexity (Rao et al., (2003), (Labrou et al., 1999).

4.5.3 Extensible Markup Language (XML)

ACLs discussed thus far, by no means cover the entire spectrum of what applications may want to exchange. A third attempt to produce an ACL makes use of XML (Bradshaw et al., 1999). XML is a language for creating markup languages that describe data. It is a machine readable and application-independent encoding of a "document" e.g., of a FIPA-ACL message including its content. XML organizes data by using tags to classify data into parts created by a document type definition, which can be authored and interpreted without ambiguity by other agents or by applications receiving the XML structured data Chaib-Draa (2002), (Grosof et al., 1999). According to Tauber (1999) and Rao et al., (2003), XML besides being easier to encode, is more "WWW-friendly" and has a much wider and more flexible way of defining base ontologies compared to KQML or FIPA-ACL.

The following section presents the evaluation factors that have been developed with respect to agent communication to be used in evaluating the mobile agent systems. These criteria are based on the discussions presented in this chapter and are used in chapter 6 for mobile agent system evaluation.

4.6 Evaluation Factors

1) Nature of message passed (Synchronous / Asynchronous):

2) In the case of synchronous message passing-
   Mechanisms supported (Message passing / RPC / Future reply/ Streams/ Events / Tuple Space):

3) In case of asynchronous message passing-
   Mechanisms supported (Message passing / RPC / Future reply/ Streams/ Events / Tuple Space):

4) Nature of information exchanged (Strings / Objects / Raw-bytes):
   This factor helps evaluate the format of the information that is exchanged between the parties.

5) Manner in which communication parties are addressed (Direct / Indirect):

6) Mechanisms for Agent - Agent communication:
   a) Intra-place: Message passing / RPC / Future reply/ Streams/ Events / Tuple Space
      This factor is used for determining the communication mechanisms between two agents located at the same place.
   b) Inter-place: Message passing / RPC / Future reply/ Streams/ Events / Tuple Space
      This factor is used for determining the communication mechanisms between two agents located at different places.

7) Mechanism for Agent - Service Agent communication:
   a) Intra-place: Message passing / RPC / Future reply/ Streams/ Events / Tuple Space

This factor helps in determining the communication mechanisms used between an agent and a service agent when they are located at the same place.

b) Inter-place: Message passing / RPC / Future reply/ Streams/ Events / Tuple Space

This factor helps in determining the communication mechanisms used between an agent and a service agent when they are located at different places.

8) Mechanisms for Agent – Agent group communication:

a) Intra-place: Tuple Space / Blackboard

This factor determines the communication mechanisms used between an agent and a group of agents when they are located at the same place.

b) Inter-place: Tuple Space / Blackboard

This factor determines the communication mechanisms used between an agent and a group of agents when they are located at different places.

9) Agent identification scheme (Global Unique ID / Based on Host Names / name – arbitrary String)

This factor helps in evaluating the agent identification schemes to determine if the agent is identified by some globally unique identity generated by the system itself or is the name given to the agent derived from the name of the host on which it is executing or is it just a name give to the agent randomly?

10) Name unchanged during agent's life (Yes / No):

This factor helps evaluate the agent-naming schemes to determine whether the identity of the agent remain constant throughout an agent's life or does it change as agents migrate.

11) Support for location transparency at application layer (Yes / No):
This factor helps in evaluating the location transparency mechanisms to determine whether the mobile agent systems provide means for abstracting the agent in a location independent manner.

12) Mechanisms for locating agents (Probing / Logging / Advertising / Brute force / Path of proxies):

13) Naming mechanisms for places (DNS / URL):

14) Name resolution mechanisms (DNS / Directory services / Table lookup)

15) Compliant to ACL (Yes/ No):
This factor helps in evaluating the MAS to determine whether it supports any of the agent communication language.

16) Support of communication in heterogeneous languages (Yes / No):
This factor examines the MAS to determine the possibility of communicating in a heterogeneous language.

17) Mechanisms to support heterogeneous in communication language (AIDL / different interpreters):

4.7 Conclusions and summary

From the analysis of the information presented in this chapter, it is suggested that the mobile agent systems should support both synchronous and asynchronous message passing. The exchange of objects rather than just string is recommended for information exchange between agents as more meaningful information can be associated it. When agents address each other for communication purposes, use of indirect addressing mechanism (e.g. use of a proxy) is recommended as this could provide an improved security. For communication between agents/service agents on

different places, the suggested mechanism to use is message passing as the overhead involved is less when compared to, for instance RMI.

With regards to the agent naming mechanisms, global unique names that remain constant and independent of the host name is suggested as this improves ease of locating the agents. Use of distributed database for logging the location of mobile agents is suggested as this alleviates the problems associated with centralized databases.

The ability of agents systems to communicate in heterogeneous languages is recommended, and this can be facilitated by the use of XML. Thus, compliance to an ACL is suggested.

This chapter contains an overview of the issues involved in agent communication. Sections 4.2 and 4.3 present mechanisms involved in identification and location of agents whereas sections 4.4 and 4.5 presented an analysis of the types of communication and the mechanisms involved in those types of communications. Section 4.6 presented the frameworks to facilitate interoperability of agents in heterogeneous environment. In section 4.7, evaluation factors based on agent communication issues were developed. The next chapter examines the security related issues involved in mobile agent systems.

5 Agent Security

5.1 Introduction

From the information presented in this dissertation thus far, one can easily conclude that mobile agent can be a powerful tool, but if adequate security is not in place it could be a dangerous one too. In fact, security (or rather lack of it) is often quoted as a major obstacle to the widespread use and adaptation of this new technology (Johansen 1999), (Farmer et al., 1996), (Jansen 2000), and (Tripathi et al., 2001).

In any distributed system, whenever a request for a certain service is received, the receiving principal needs to address at least two questions: Is the requesting principal the one it claims to be, and does the requesting principal have appropriate privileges for the requested services? These two questions refer to the issues of authentication and authorization (Lee et al., 2004), (Vigna, 1998). There are other security concerns such as auditing, secure communication, availability and accountability that need to be addressed. When it comes to mobile agents, the security issues become further complicated given that there are greater opportunities for abuse and misuse. Compared to the client-server model, mobility in mobile agent model increases the threat of security violations, due to possible dynamic injection of malicious agents by untrustworthy users that can compromise the hosting node's resources in a way similar to that of viruses and worms. In addition, there exists new security problems specific to mobile agents, for instance, a host entrusted with the responsibility of agent execution could try to tamper with the agent code and state, with an intent to disclose private information and to block agent transfer to successive execution sites (Johansen, 1999) and (Orso et al., 2001).

In section 5.2 a security model to categorize the threats to a mobile agent system is presented. Thereafter, section 5.3 and 5.4 contains an analysis of these threats as they apply to various components of the mobile agent system. In addition, the sections contain an outline of the key features and point out the limitations of the common protection techniques suggested to protect both agent hosts and the agents against reciprocal malicious behaviors. In section 5.5, the requirements for providing protection to MAS are presented. Based on the issues discussed in sections 5.1 to 5.5, section 5.6 presents the factors to be considered in evaluating the MAS. Section 5.7 presents the conclusions and summary.

5.2 Security Model

A number of models exist for describing a mobile agent system. One such model is as described by Vitek and Tschudin (Vitek et al., 1998) which consists of four components: A host which is a computer with its operating system, a computational environment (CE) or agent execution environment (AEE) which is the runtime system, an agent, and the network or the communication subsystem that interconnects CEs located on different hosts. Another such model is as presented by Jansen (2000). According to this model, shown in figure 13 below, a mobile agent system consists of only two components: agent and agent platform that provides the computational environment in which an agent operates. An agent platform may support multiple locations or meeting places where agents can interact.

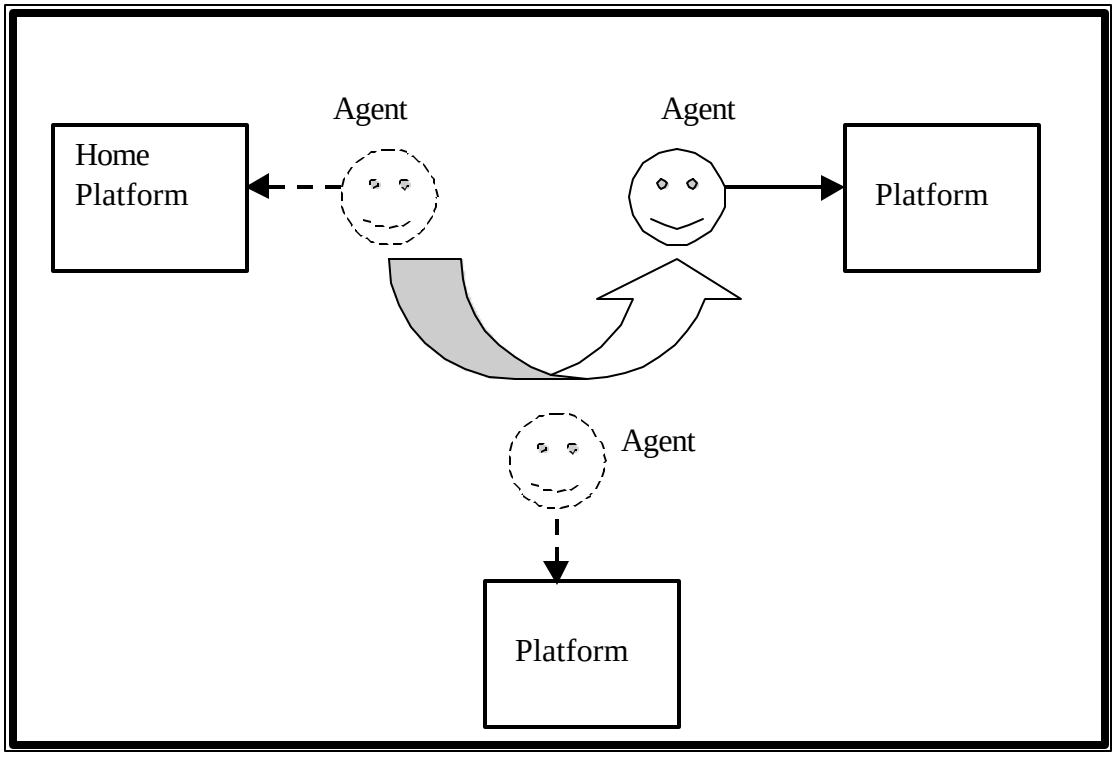

Figure 13: Agent System Model (Jansen, 2000)

To facilitate the discussion on security issues, a slight modification to Jansen's model is made. The proposed model consists of *agent* and the *host* as its major components. Since the platform (or platforms) is contained in a host, providing adequate protection for the host will also provide protection for the platform. For example, protecting host from attack by malicious agent will also protect the platform from attack by the agent. Using this modified model, providing security for mobile agent system could be divided into two broad categories of providing security for the agent and providing security for the host. Providing security for the agents can be further subdivided into protecting the agent while it is at a host and while it is in transit. The agent at a host could be attacked by the host or by other agents co-located at the host. Meanwhile, the host security should cater for protection of the host from various forms of attack an agent can lounge when visiting the host and also attacks from other hosts in the system. The above discussion is depicted in Figure 14 below.

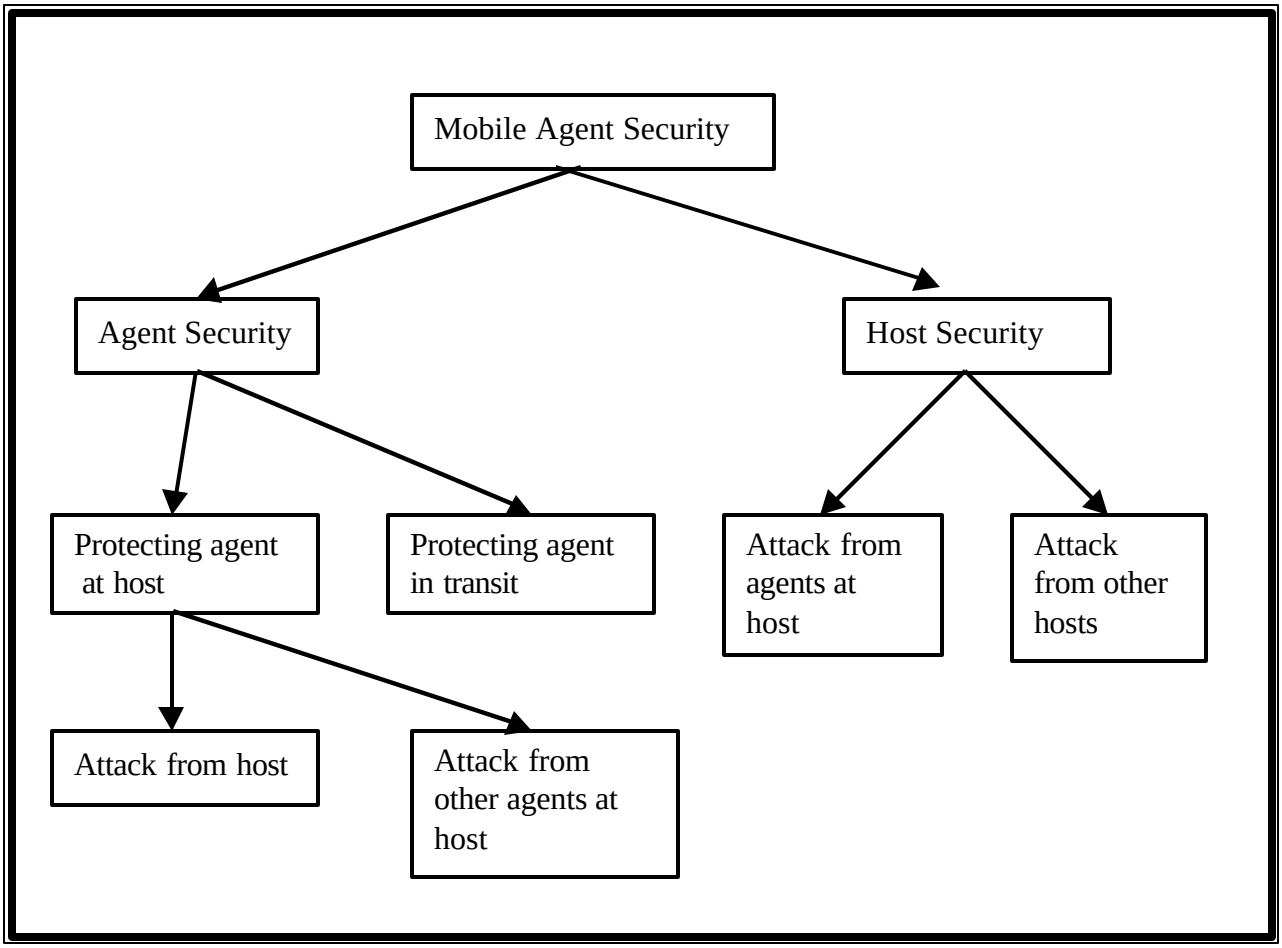

Figure 14: Mobile Agent System Security.

In addition to the various attacks depicted in figure 14 above, against which adequate protection must be provided, some researchers also consider attack of the host from execution environment and vice-versa. According to Pham et al. (1998), protecting the host from the agent execution environment and the execution environment from the host is similar to the traditional problem of protecting an operating system from misbehaving programs and vice-versa. This is not a new problem and is less important in the context of mobile agent security hence it will not be analyzed any further. This supports the reasoning for the proposed model and hence does not consider the platform as a component for analyzing security concerns.

5.3 Agent Security

Providing for agent security involves protecting the agent while it is stationed at a host as well as when it is migrating from host to host in order to fulfil its tasks. In order to provide adequate security for the agents, it is important to analyze the kinds of attack that can be launched against them. In this sub-section and the next, the threats that agents may be exposed to while at a host are analyzed. It must be noted that the threats mentioned below, with respect to agent and host security, are by no means exhaustive. They are intended to show the gravity of the problems faced by the implementers of mobile agent systems.

5.3.1 Attack from Host

As mentioned earlier, a mobile agent migrates from one host to another with its code, data and state. As the host provides the necessary execution environment for the visiting mobile agents, it has almost complete control over the agent. This provides a malicious host with the necessary opportunities to initiate several types of security attacks on the mobile agent. The following section presents the types of attack faced by agents from their hosts and are based mainly on the works presented by Biermann & Cloete (2002), Montanari et al., (2001), Jansen (1999), and Jansen (2000).

One of the ways in which an agent can be attacked by its host is by having its privacy invaded. This is generally performed with the aim of gathering or stealing information in order to benefit from the agents operations (Pagnia et al., 2000). A host could easily corrupt an agent by deleting or altering the agents code, data, flow control or status. The host can also compromise the agent's integrity by interfering with its intended mission or responding falsely to requests for information or services, or diverting it to another platform not on the agent's itinerary. In the worst case, the host may even kidnap the mobile agent and prevent it from ever returning to it home machine or create its own similar agent for use in an unfair manner by making use of reverse engineering principles.

A host can masquerade as another host in an effort to lure an unsuspecting mobile agent to it in order to extract sensitive information from these agents. The masquerading platform can harm both the visiting agent and the host whose identity it has assumed. Cloning of an agent together with masquerading creates authentication problem.

5.3.2 Attack from other agents at the host

In many mobile agent system architectures, system-level services such as directory services and inter-platform communication services are provided by components that are agents themselves (known as stationary agents). A visiting mobile agent, in search of a particular service could expose itself to various forms of attack from these stationary agents or other mobile agents sharing the same execution environment. For example, a stationary agent could delay or prevent a visiting agent from accessing one or more resources or services available at the host. Malicious agents can also intentionally distribute false or useless information to prevent other agents from completing their tasks correctly or in a timely manner.

To accomplish their tasks, agents have to communicate directly with one another. An agent may attempt to disguise its identity in an effort to deceive the other agent into releasing sensitive information to which it is not entitled. For example, an agent may pose as a well-known dealer of particular goods or services and try and convince the other unsuspecting agent to provide it with credit card numbers.

As described by Xudong et al. (2000), another form of attack which one agent can perform on another agent is repudiation. Repudiation occurs when an agent participating in a transaction or communication later claims no knowledge of it ever having taken place. A proper countermeasure must be in place to resolve this as it could lead to serious disputes that may not be easy to resolve.

When an agent host has a weak or no control mechanism in place, several other forms of attack are possible between the agents. An agent can directly access or modify an agents' code or date thereby radically changing the agents' behavior or property. An agent may also gain information about the other agents' activities by using host services to eavesdrop on their communications.

Having analyzed the security concerns of an agent, the section below present ways for reducing these concerns.

5.3.3 Protection of the agents

Protecting mobile agents from the host is near impossible task (Lee et al., 2004), and (Vitek et al., 1997). The host must be able to read the complete agent code and data in order to execute it. The host is able to inspect - at any time of execution - the complete run-time stack and may do any modification it wishes to. The agent is at the mercy of the system. The directions taken in dealing with a hostile host is addressed by a few proposals, which aims either to prevent or to detect attacks (Sander et al., 1998), (Vigna 1998).

Prevention mechanisms try to make the spying or tampering of agent code/state impossible. One of the ways this can be achieved is as suggested by (Lee et al., 2004) and (Farmer et al., 1996) where mobile agents are only able to circulate in trusted execution environments. This approach does not support the loose roaming itineraries necessary in many agent applications. Another way would be to execute the agents in a physically sealed environment. This is achieved by exploiting temper-proof hardware that prevents even the administrator of the device from accessing the agents (Chan et al., 2002), (Wilhelm et al., 1998). However, their deployment is low, mainly because of the high costs involved.

Prevention of agent tampering that is software-based, is a relatively new field of research (Lee et al., 2004). Most of them, as suggested by Hohl (1998) and Montanari et al. (2001), make use of either the scrambling techniques to encrypt certain portions

of the agent or the obfuscation techniques. The strategy behind the scrambling techniques is to scramble the code in such a way that no one is able to gain complete understanding of its function, or to modify the resulting code without detection. A serious problem with the general technique is that there is no known algorithm or approach that provide a Blackbox property (Orso et al., 2001), (Schelderup et al., 1999). An agent is said to have a Blackbox property if its code and data cannot be read or modified. Another solution making use of encryption is as suggested by Sander et al. (1998). In this case, the agent is not the code that is encrypted but the function this code executes. The major challenge here is to find encryption schemes for arbitrary functions.

The obfuscation technique approach could be useful in certain occasions where an agent carries time-limited token-data. However the security of this method cannot be proven (Montanari et al., 2001) and the technique lacks approach for quantifying the protection interval provided by the obfuscation algorithm, thus making it difficult to apply in practice.

Detection mechanisms are employed to assist in the detection of any illegal modification to agent code and/or state that might have occurred while an agent was executing at a host. They may also assist in identifying illegitimate hosts and prove their misbehaviour. One such mechanism is the tracing mechanism that was introduced by Vigna (1998), which records the execution of the agent and its interaction with the execution environment. Yi et al. (1998) proposed the use of Agent Service Centre, which traces the itinerary of the agent. In a similar attempt, (Page et al., 2003), (Kotzanikolaou et al., 1999) make use of multi-agent system that can trace which mobile agents were victims of malicious behaviour. This approach has a number of drawbacks, the most obvious being the size and number of logs to be retained, and the fact that it is possible to detect tampering with agent execution, and determine which target host was the perpetrator only after the agent has finished its task and returned to the originator. Other more subtle problems identified include the lack of accommodating multi-threaded agents and dynamic optimisation techniques (Jansen et al., 2000). While the goal of this technique is to protect an agent, the technique does afford some protection for the agent host, provided the host can also obtain the relevant trace summaries from the various parties involved.

The most common approach taken for protecting the mobile agent during the period it is travelling between the hosts is by making use of various techniques of encryption on agent's code and data (Gray et al., 2002). While encryption attempts to provide privacy, digital signatures are used to verify that the agent has not been tampered with during transit. It must be noted that the actual transportation of the agent is achieved by making use of transport protocols such as HTTP or TCP/IP. Some agent's transport protocol such as Secure Socket Layer (SSL) and Transport Layer Security (TLS) also provide security to the agents being transported. A better alternative is to use the Key Exchange Protocol (KEP) as it offers a lightweight transport security mechanism (Gray et al., 2002).

5.4 Host Security

A mobile agent paradigm requires an agent host to accept and execute an agent. Without adequate defense mechanisms, an agent host is vulnerable to attacks from malicious agents and other entities such as other malicious hosts in the system. In order to provide adequate security for the host, one has to consider the types of attack the entities mentioned above can execute against the host. The following section contains an analysis of the attacks that a host is vulnerable to from the agents visiting it and from other hosts in the system.

5.4.1 Attack from agents at the host

A malicious mobile agent can change or destroy resources or services by modifying, reconfiguring, or erasing them from memory or disk. When a mobile agent deliberately damages a mobile agent host, it could potentially damage all other mobile agents executing there at the time Borselius (2002), (Greenburg et al., 1998). Examples of destruction include deleting or writing randomly into files, ordering an

unscheduled hardware upgrade to a host, or modifying of the system configuration to change the security policies.

In an environment with inadequate control, an agent can access and steal private information, for example, secretly recording a host's communication and then transmitting it over a network to an unauthorized site using covert channels to steal data in ways that violates a host's security policy (Jansen, 2000), (Kaufman, 1995). Another form of attack that an agent can execute against the host is to block one of the processes by overloading its buffers to create a deadlock and hence preventing its use Li et al., (2004). Masquerading is yet another form of attack that an agent may execute on the host. The masquerading agent may assume the identity of another agent in an effort to gain access to services and resources to which it is not entitled.

5.4.2 Attack from other hosts in the system

Assuming that the agents on a host are well behaved, it is possible for a host to be attacked from other entities such as another host in the system in an attempt to disrupt, harm, or subvert the agent system. According to Wallin (2004), this category of attack is focused largely on exploiting the vulnerabilities of the communications capability of the host. A host may attack the inter-agent or inter-host communication through intercept, forgery or replay. For example, a host may intercept a message in transit between agents or hosts to gain information. This information can then be modified, substituted with other information or simply replayed at a later stage in an attempt to disrupt the synchronization or integrity of the agent framework.

Having analyzed the security concerns of a host, the section below present ways for reducing these concerns.

5.4.3 Protection of the Host

When a host is executing an agent's instructions it must take precautions so as not to open itself to hostile attack from malicious agents. Many of the traditional security techniques used in contemporary distributed applications can be utilised for providing protection for a host in a mobile agent paradigm. Many of these techniques are based on information fortress model (Sameh et al., 2002), (Blakley, 1996), where a closed system accessed through well-defined and regulated interface is maintained.

Another approach to reduce the vulnerability of the host is to load the suspected code into its own part of address space known as fault domain or sandbox. The code is then modified to make sure that each load, store, or jump instruction is to an address in the fault domain (Appel et al., 2000), (Gong 1997). However, the rigidity of this approach makes it inadequate for complex agent-based application and the code is no longer platform-independent. A similar approach to sandbox, used by Safe-Tcl is called the padded cell. Padded cell isolates the agent in a safe interpreter where it cannot interact directly with the rest of the application. In turn, a trusted master interpreter controls the execution environment of the safe interpreter. A disadvantage of using interpreters is the serious performance overhead involved when compared to a compiled machine code.

Making use of safe programming languages can also reduce vulnerability of the host. A safe programming language can enhance safety by enforcing strong typing, restricted memory-reference manipulations, and runtime-supported memory allocation and de-allocation (Montanari et al., 2001).

Theft or damage of sensitive data and denial-of-service attacks are prevented by making use of access control policies in conjunction with mechanisms to limit the rate and amount of resource consumable by an agent at a host (Gray et al., 1998), (Feridun & Krause 2001), (Gray et al., 2002), (Picco 2001), and (Tripathi et al., 2001). In the application of chapter 2, agent's access control is implemented by configuring the `aglets.policy` file. Figure 15 below gives a screen shot of a section of the policy file on host Jupiter.

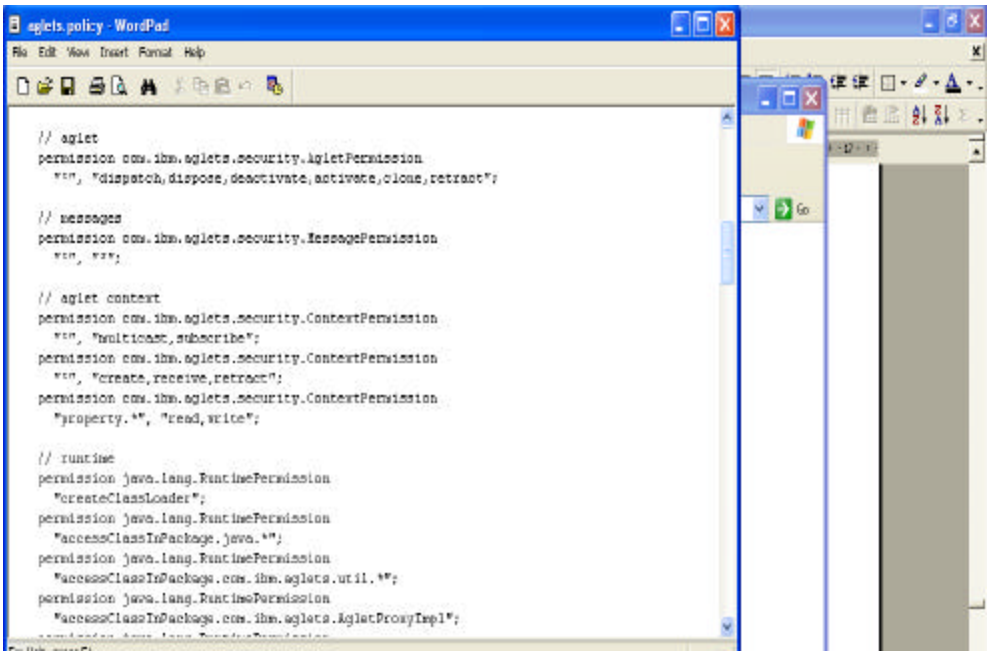

Figure 15: Screen shot of `agents.policy` file

It can be seen from the screenshot above that an agent on this host has the permission, for example, to dispatch, dispose and clone other agents. Similarly, permission has been assigned to all agents on this host to send all kinds of messages (e.g. `message.show`).

The host must be able to authenticate the agent's owner, assign resources based on this authentication, and prevent any violation of those resource limits. Authorization and authentication could be achieved by making use of digital signatures and cryptographic techniques. To prevent execution of unsafe code, Proof Carrying Code technique is suggested (Appel et al., 2000), (Necula et al., 1996). In this technique, the agent's code is compiled on the home machine together with a proof that the agent satisfies a given security policy. The binary code and the proof can then be sent to host machines. The host machine validates the proof and then can safely execute the agent. The proofs and the code generated by this technique has the property that if the code or the proof is tampered, then on arrival at the destination host, either the proof will not validate the code and be rejected, or it will validate the code but the code will

not be able to compromise the host security (Pleisch et al., 2003), (Lee et al., 1997). Using Proof Carrying Code, a host can avoid not only the overhead of sandboxing, but also some of the policy-checking overhead in using Safe-Tcl for achieving system security. Like the basic sandboxing technique, Proof Carrying Code sacrifices platform-independence for performance.

In a recent attempt to address the authentication and authorization concern, a XML-based Security Association Markup Language (OASIS, 2003) makes use of a single sign-on that permits an authenticated user of one domain to use resources from another domain without the need for re-authentication. With it, a security administrator can express advanced security requirements, such as time or event-based restrictions (Schoeman & Cloete, 2003). It must be noted that authentication credentials do not guarantee that the mobile agent will be harmless.

5.5 Requirements for protecting the Mobile Agent System

While agents undoubtedly provide many advantages, the design of a secure mobile agent system has been hampered by the lack in ability of the current systems to provide adequate protect, in particular, to the mobile agents from being tampered by a malicious host (Biermann et al., 2002). From the various research performed by researchers such as Weyner (1995), Jansen (1999), Biermann & Cloete (2002) and from the discussion above, many mobile agent security issues can be identified. These security issues, in a broader context, can be translated into a set of security criteria that a mobile agent system needs to fulfil. This can be summarized as follows:

Confidentiality of the system: The system must provide mechanisms for secure communication and secure transfer of agent components as it migrates from host to host in an insecure network.

Integrity of the system: There must be mechanisms in place to detect tampering with the agent system.

Authentication of the various entities of the system: The entities such as the agents and hosts participating in the mobile agent applications must be unambiguously identifiable.

Authorization and access control: The agent system must have some mechanisms in place to protect their resources by specifying their access control policies and be in a position to enforce them.

The following section presents the factors that have been developed with respect to agent security to be used in evaluating the mobile agent systems. These factors are based on the discussions presented in this chapter and are used in chapter 6 for mobile agent system evaluation.

5.6 Evaluation factors

1) Mechanisms supported for agent confidentiality (Cryptographic techniques):

2) Mechanisms to support agent integrity. (Cryptographic techniques):

3) Mechanisms for authentication of the agent's owner. (Digital signatures):

4) Mechanisms used for authorization and access control (access list/ policies):

5) Mechanisms to protect host:
   a) From attack by the agents collocated at the host.
   b) From attack by other hosts in the system

6) Mechanisms to protect agents:
   a) From attack by the host.
   b) From attack by other agents collocated at the host

5.7 Conclusions and Summary

From the discussions presented in this chapter, it is imperative that the agent systems provide adequate security for agents and the hosts from being attacked by each other. It can be concluded from discussion thus far that the traditional mechanisms for providing authentication, integrity, and confidentiality can also be applied to mobile agent systems with limited successes. When protection for the host from agents is needed, both padded cell and sandbox mechanism is applicable but their limitations should be kept in mind when a choice of a mobile agent system is being made. Padded cell and sandbox mechanism could also be used when protecting collocated agents from one another. The real problem that mobile agent systems are facing is the lack of mechanisms to protect an agent from attack by its host. The suggestion in this regard is to make use of appropriate hardware.

In this chapter, a study of the various kinds of threats that a mobile agent and its hosts may be exposed to is made. The analysis of these threats was important so that adequate protections against those threats could be provided. Section 5.3 and section 5.4 dealt with the kinds of attack one component can perform against the other. Section 5.5 presented the requirements of confidentiality, integrity, authentication and authorizations of the system. Finally, in section 5.6 the evaluation factors were presented. In chapter 6, MAS are evaluated against the factors presented in the previous three chapters.

6 Evaluations Of The Selected Mobile Agent Systems

6.1 Introduction

Mobile agent systems can be categorized in several ways. In this dissertation, for the purpose of evaluating MAS, a small but representative sample of MAS was selected. The selection of mobile agent systems was facilitated by grouping the MAS based on the types of mobility, communication and the languages used in their development. It is assumed that the mobile agent systems within each group have similar properties. For example, all mobile agent systems developed in a particular language with similar type of mobility and communication will have similar properties.

Although this study does not focus on the languages for agent programming, they are being considered here as they has a bearing on the ways mobility, communication and security related aspects are supported. For the purpose of this research, the agent programming languages have been broadly classified as "Java based" or "Non-Java" based systems. It is worth noting that most non-Java based systems are also multi-language systems and those multi-language systems that also support Java (such as D'Agents and Ara) have added this support at a later stage and did not form an integral part of their original design (Peine, 1997). Hence for the purpose of this dissertation are not classified as "Java based" systems. From the extensive literature survey conducted, it was found that all mobile agent systems that provided support for global communication also supported local communication.

Section 6.2 presents the grouping of the mobile agent systems from which one MAS is selected for evaluation. In section 6.3 a brief introduction to the selected mobile agent systems together with the mechanisms for mobility, communication and security of the selected MAS at implementation level are presented. In Section 6.4 the

analysis of the mobile agent systems against the evaluation factors are presented and displayed in a tabular form.

6.2 Grouping of Mobile Agent Systems

Using the types of mobility, communication and the languages as the three factors, the figure below presents a possible grouping scheme.

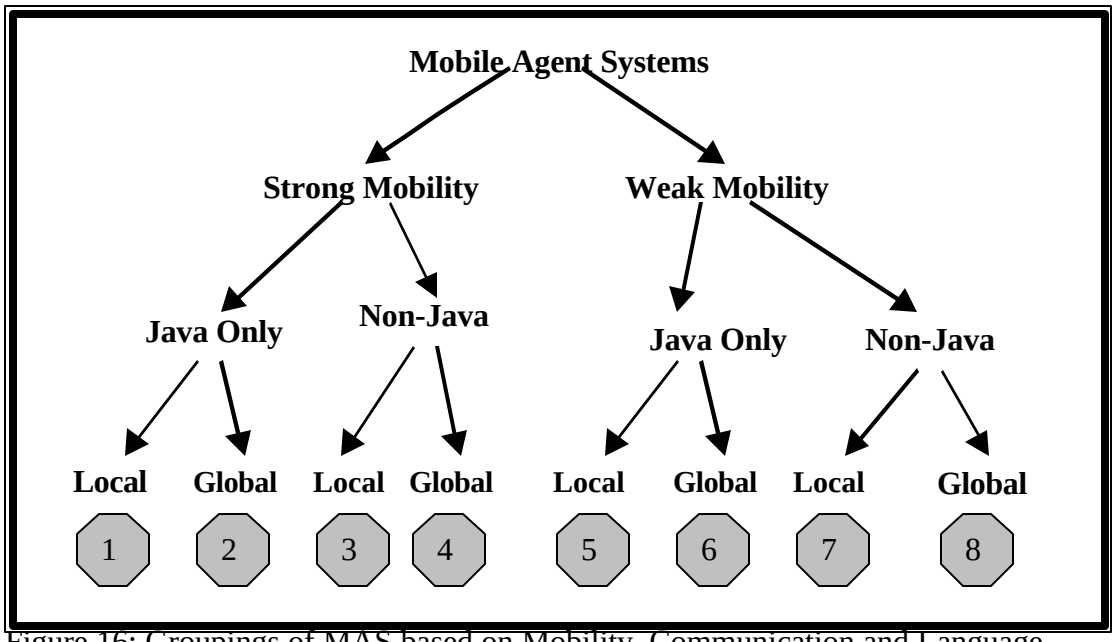

Figure 16: Groupings of MAS based on Mobility, Communication and Language.

From the above figure it is apparent that there are eight possible groups (numbered 1 to 8). Group 1 consists of all Mobile agent systems supporting strong mobility, developed in Java, and support local communication only. Similarly, group 2 represents mobile agent systems that support strong mobility, developed in Java and support global communication. By the literature survey conducted for this research no mobile agent systems satisfying the conditions of group 1 and group 7 were found. For this reason, the agent systems are grouped in six groups. The subsections 6.2.1 to 6.2.6 present a list of agent systems belonging to these six groups (groups have been re-ordered). From each group, one agent system was selected for evaluation against the factors presented in chapters 3, 4 and 5. Since the selected agent systems belong to a group with similar characteristics, it can be deduced that all mobile agent systems in

that group will have similar properties. The selection of the agent system to represent the group was based on the popularity of that system (determined by the number of scientific articles found during the literature survey).

6.2.1 Group One

This group comprises of agent systems that are non-Java based, support local and global communications and provide strong mobility. Table 2 lists the agents of this group. From this group D'Agents was selected. (It is to be noted that the mobile agent systems listed in tables 2 to 7 are obtained from the mobile agent list available at *http://mole.infomatic.uni-stuttgard.de/mal/preview.html*)

Cborg	**D'Agents**
ffMAIN	**IMAGO**
Jinni	**MiLog**
Mobiget	**Normadic Pict**
Planet	**Telescript**

Table 2: Mobile agent systems of Group 1

6.2.2 Group Two

This group represents agent systems that are non-Java based, provide weak mobility and support local and global communications. TACOMA was selected from this group. Table 3 lists the agent systems that belong to this group.

Bond	**Dejay**
dynamicTAO	**Gypsey**
Knowbot Operating Environment	**MO Messenger**
TACOMA	**TextAgent**

Table 3: Mobile agent systems of Group 2

6.2.3 Group Three

This group represents agent systems that are developed in Java, provide strong mobility and support local and global communications. Table 4 lists the agent systems of this group. WASP was selected to represent this group.

AgentSpace(UK)	**EAS**
JAM	**Klaim**
MATS	**NOMADS**
owchii	**Plangent**
Tagent	**WASP**

Table 4: Mobile agent systems of Group 3

6.2.4 Group Four

Represents agent systems that are developed in Java, provide weak mobility and support local and global communications. Amongst the groups, this group has the most number of members as shown by the table 5 below. Aglets are selected to represent this group.

AgentSpace	**Ajanta**
Aglets	**AMASE**
aIsland	**AMETAS**
Anchor Toolkit	**ARCA**
aZIMAS	**Bee-gent**
Concardia	**DIAT Agent**
erwerywer	**FarGo**
Grasshopper	**Hive**
IMAJ	**J-SEAL2**
JAE	**JAMES**
JavaNetAgents	**JavaSeal**
JCAFE	**Jumping Beans**

Kaariboga	**MAgNET**
MAP	**MILLENNIUM**
MIPLACE	**MOA**
MobiliTools	**Mogent**
Mole	**MuCode**
Odyssey	**Pathfinder**
SeMoA	**SOMA**
Tracy	**VoyagerORB**

Table 5: Mobile agent systems of Group 4

6.2.5 Group Five

This group comprises of non-Java based mobile agents that support strong mobility and implement local communications only. This group has just two members with ARA representing the group.

ARA	**MESSENGERS**

Table 6: Mobile agent systems of Group 5

6.2.6 Group Six

This group comprises of MAS that are developed in Java, support weak mobility and implements local communications only. This group has only one member as indicated by table 7 below.

RMI64	

Table 7: Mobile agent systems of Group 6

RMI64 is a simple mobile agent system consisting of just 3 java classes. The sole purpose of this system was to show how simple it is to write a minimalist mobile agent system using Java. The only capability of an agent in this system is its ability to

migrate (weak migration) to another host and begin its execution there. The RMI64 mobile agent system does not provide any kind of security whatsoever nor does it have any mechanisms in place to deal with issues such as agent naming, name resolution or providing support for agent persistence. It only supports local communications with RMI being its only mechanism for communication and transportation (Gschwind, 1999). Based on the discussion presented in this paragraph, RMI64 is not considered as a full-fledged mobile agent system and being the only one in this group, the group is not considered for further evaluation. This leaves five groups of mobile agent systems for evaluation against the factors. Section 6.3 below presents an analysis of the agent systems.

6.3 Analysis of the Selected Mobile Agent Systems

A brief introduction to the selected mobile agent systems together with the mechanisms for mobility, communication and security of the selected MAS at implementation level are presented below.

6.3.1 D'Agents (Version 2.1)

D'Agents (formally known as Agent Tcl) was developed at Dartmouth Collage to address the weaknesses of then existing mobile agent systems, such as insufficient security mechanisms, difficult or nonexistent communication facilities and inadequate migration facilities (Gray, 1995). It is developed on Unix and can be written in multiple languages such as Tool command language (Tcl), Java or Schema. One of the design goals of this system has been the support for strong mobility (Gray et al., 2002). In order to support strong mobility in Java means that the JVM has to be modified, leading to a situation where this modified JVM run the risk of soon becoming incompatible with the later versions of the JVM.

In D'Agents, a EU (called agent) is a Unix process having its own separate address space and share only resources, considered to be non-transferrable, provided by the

underlying operating system. The CE abstraction is implemented by the operating system and the language run-time support (Fuggetta et al., 1998). One of the key components of D'Agents architecture is the agent server that runs on each host. The server provides communication, migration and bookkeeping services to the agents residing on its host. The server is multi-threaded; each agent (Tcl and Schema) is executed in a separate process, which simplifies the implementation considerably, but adds the overhead of inter-process communication (Gray et al., 2002). Java agents are executed as separate threads inside a single Java process. More specifically, the server starts a small number of Java Virtual Machines (JVMs), and executes multiple agents inside each. D'Agents approach of multi-threaded server, but separate interpreter processes for the agents, allows both high performance and straightforward support for multiple languages.

In D'Agents, the site to which an agent migrates is addressed by their DNS names and utilizes TCP/IP as the primary agent transportation mechanism, although other mechanisms such as e-mails are supported. According to Gray (2002), D'Agents do not have any mechanism to support agent persistence when the host machine crashes. D'Agents support both static and dynamic itinerary.

6.3.1.1 Mobility

As mentioned previously, D'Agents support strong mobility for Tcl and Java with significant modification to their interpreters. There are three migration related commands supported by this system (Gray, 1996).

The `agent_submit` function pushes the code component of an EC to another CE. This function takes as arguments the machine name and the code. Additional parameters such as variables and procedures of the parent agent that a child agent may need are also passed. The variables are passed by copy.

`agent_submit` machine (*-procs* proc_name) (*-var* var_name) *-script* script

The `agent_jump` function migrates the EU to another CE. This function takes the machine name as a parameter and optionally a time indicating the maximum number of seconds to wait for a response from the destination CE to indicate its success.

`agent_jump` machine (-*time* sec)

The third command is the `agent_fork` function, which is analogous to Unix fork. The command creates a copy of the agent on the specified machine. The child and the parent continue their execution from the point at which fork occurred. As in the `agent_jump` command, this command too takes time as on optional parameter.
`agent_fork` machine (-*time* sec)

Both `agent_jump` and `agent_fork` are proactive migration mechanism enabling strong migration and the bindings in the data space of the migrating EU are removed. `agent_submit` migration mechanism is asynchronous and immediate with only the code being migrated. The strategy used for code relocation is heavyweight, with all the classes needed by the agent are moved at the time of agents migration. In all the three migration primitives mentioned above, the migration is absolute - name of the destination machine is essential for the migration. D'Agents do not have any mechanism to allow one agent to migrate another (Gray et al., 1998).

6.3.1.2 Communication

D'Agents server provides a per-host namespace for agent communication. Each agent is identified by a combination of the network address of the machine on which the agent is currently residing, a unique integer that the machine's server assigns to the agent when it first arrives and optionally a string name. The last two are unique only to the current machine (Gray et al., 2001). In a similar fashion, the places identity is derived from the host address. For the purpose of name resolution, DNS services are utilized. Within this namespace, the following mechanisms for both local and remote communication could be used (Gray et al., 2001).

6.3.1.2.a Message passing

In this mechanism, agents exchange arbitrary strings or binary data with standard `agent_send` and `agent_receive` primitives. The format is as shown below:

`agent_send` *machine integer string* (*-time* sec)

integer is numeric code, *string* is the message, *machine* is the destination machine, and *sec* is an optional parameter indicating the time in seconds to wait for a response from the destination machine.

`agent_receive` is used to receive a message. The command has blocking, nonblocking and timed forms. The blocked form waits until a message is available and then sets the variable *code_var* to the message code, sets the variable *string_var* to message string, and returns the identity of the sender.

`agent_receive` *code_var string_var* (*-blocking* | *-time* sec | *-nonblocking*)

The nonblocking form returns -1 if no message is available. Otherwise, it sets the *code_var* and *string_var* to the message code and string respectively and returns the identity of the sender.

6.3.1.2.b Streams

Using the streams mechanism, agents establish a direct connection with a target agent and then send string or binary messages across the connection.

6.3.1.2.c Events

Events are used for asynchronous notification of an occurrence of a particular incident. Using the events mechanism, an agent sends events to another agent, which catches and processes the events with registered event handlers.

6.3.1.2.d Restricted form of RPC

D'Agents provides its own variant of RPC called ARPC (Agent RPC), with its own Agent Interface Definition Language (AIDL), an IDL compiler and directory agents where interfaces to the services can be registered. ARPC is available only to agents and server agents written in Tcl language, so that no parameter marshalling and related interface conversions are necessary (since all data are simple text strings in Tcl). It is for this restriction that ARPC are generally used for communication between agents on the same host (Gray et al., 2001).

All these mechanisms are implemented on top of TCP/IP. As the agents migrate from one host to another in order to fulfil their tasks, they leave behind a pointer to the next destination, which is used to locate an agent. In addition to leaving behind a list of pointers, D'Agents also have facility for logging an agent's location on databases that can be quarried to determine the agent's location. D'Agents make use of the Agent Interface Definition Language to allow agents created in different languages to communicate using approach similar to CORBA IDL (Hooda et al., 1998)

6.3.1.3 Security

The design of D'Agents, focuses mainly on providing protection for the host machine. Each D'Agents server distinguishes between two kinds of agents: owned and autonomous. An owned agent is an agent whose owner could be authenticated and is on the server's list of authorized users whereas an autonomous agent is one that cannot be authenticated or is not on the server's list of authorized users. Based on this distinction an agent is assigned access to the resources (Gray et al., 2002).

Agent authorization functionality in this mobile agent system is distributed between the respective language interpreters and language-independent system components (e.g. resource management agent). The interpreter uses its native language mechanisms to manage and enforce access rights that are implemented as lists, mapping the requested resource types to available quantities. The interpreter in consultation with resource management agents obtains the access rights for a specific agent. The resource manager agents are a standard part of D'Agents system and are associated with specific resources through a list of access rights for each agent (Gray et al., 2001)

The D'Agents servers make use of public key cryptography to authenticate the identities of incoming agent's owners as each machine and owner has its own public-private key pair. Pretty Good Privacy (PGP) is used for digitally signatures and encryption. An agent may choose to use encryption and signatures when it migrates or sends a message to another agent. If interception is of no concern then encryption may be turned off. If an agent is not concerned with tampering during migration and can accomplish its task as an autonomous agent, it turns off signatures. Similar decisions may be taken when sending messages. Turing off either signature or encryption improves performance due to the slowness of public-key cryptography (Gray et al., 1998).

Like many other MAS, such as Mole and Voyager, D'Agents do not as yet have any means of protecting agents from attacks by malicious hosts. By executing each agent in its own interpreter, a mobile agent can be protected from attack by another agent on the same host.

6.3.2 TACOMA 2.0 (Tromsø And COrnell Moving Agents)

TACOMA 2.0 (also referred to as TAX 2.0) is one of the later versions of TACOMA with a drastically different architecture from the earlier versions. For discussion purposes in this dissertation, it will be referred to simply as TACOMA from this point on. The TACOMA project focuses on operating systems support for agents and the way agents can be used to solve problems traditionally addressed by other distributed computing paradigms, e.g. the client/server model. In TACOMA the agents are modelled as a migrating process that moves through the network to satisfy client requests (Johansen et al., 2002b). TACOMA, like D'Agents, is also developed on Unix platform and can be written in multiple interpreted languages such as Tcl, Schema, Python, Perl and C. A TACOMA agent is considered to be a set of modules conceptually interchangeable and reusable (Lauvset et al., 2002).

TACOMA is a MAS providing weak mobility and supporting several different languages through the notion of virtual machines (VMs). Different virtual machines runs as separate processes and are protected from each other through memory protection provided by the operating system. It is the responsibility of the virtual machines (VMs) to execute the code in a safe and secure manner. In TACOMA, a EU is called an agent and is implemented as Unix processor running the interpreter. The Unix operating system and interpreters implement the functionality of CE (Sudmann et al., 2000).

Entities such as agents, service agents and places where the agents meet bear a globally unique identity derived from DNS names of their hosts, optionally extended by a local name to distinguish more than one place on a host. TACOMA makes use of briefcase for transporting code segments. A briefcase is essentially a collection of folders, each containing an ordered lists of elements. An element is an un-interpreted sequence of bits and forms the most basic data type in TACOMA.

Agents running on different virtual machines need to be able to communicate. The basic TACOMA library and the briefcases produced by it assure that agents can send data and receive data in a heterogeneous environment. Apart from this, it is the

function of the firewall, which is a multi-threaded process, to assist an agent to obtain knowledge of other agents running on different VM locally. The VMs may manage some resources by themselves, like CPU and memory, subject to constraints imposed by the firewall. However, to manage arbitrary resources properly, TACOMA makes use of service agents. This allows resource allocation mechanisms to handle requests regardless of which VM the requesting agent is running on (Lauvset et al., 2001a).

6.3.2.1 Mobility

TACOMA supports shipping of stand-alone codes to a destination CE by mechanisms providing both synchronous and asynchronous immediate execution. Mobility in TACOMA are achieved by the following two primitives:

The `go()` primitive communicates with a remote virtual machine and arranges for it to be built and execute the agent. If the remote virtual machine was successful in building the agent, the `go()` terminates the local agent. The `go()` command names among its parameters an agent on the destination host and the briefcase.

The `span()` primitive does the same, except that it creates a new agent with a different instance number, which is then reported back to the calling agent in a similar fashion to Unix `fork()` system call.

The strategy used in TACOMA for code relocation is lightweight, with only the base classes moved at the time of agents migration and the rest are obtained as the need arises (Lauvset et al., 2001b). Upon migration, data space management by copy can be used to provide the new EC with the resources present within the source CE cabinet. For the transportation of the agents, protocols such as TCP, HTTP and SMTP are supported. TACOMA agents have the ability to dynamically decide, based on its current state, the next host to be visited or follow the path assigned to it at the time of its creation. Agents use rearguard or replication and voting mechanism for saving their internal states to disk for potential later restoration. Provision for persistence is done at the system level. Like D'Agents, TACOMA agents do not have the ability to request migration of another agent (Sudmann et al., 2001).

6.3.2.2 Communication

Communications between different virtual machines pass through a local firewall. The firewall acts as a reference monitor and mediates all local communication between agents, as well as communication to remote firewalls and agents on remote machines. TACOMA provides two basic communication primitives, called `bcSend()` and `bcRecv()`, which are used by virtual machines to communicate with the firewall. On top of these functions the TACOMA library offers functions like `activate()`, `await()` and `meet()`, which are used for synchronization and communication. Basically, `activate()` is equivalent to a `agent_send`, `await()` is equivalent to blocked `agent_receive` and `meet()` is a RPC of the D'Agents mobile agent system presented earlier (Sudmann et al., 2000). Firewalls are also used for the purpose of locating agents, since the virtual machines need to register and deregister agents running inside them with the firewalls.

6.3.2.3 Security

TACOMA uses operating system features to encapsulate agents and provides additional security using language specific features. TACOMA also provides access control on a per agent basis. This access control is based upon the authentication of the principal whom the agent is working for (the owner of the agent). The access control in the form of vector of access right to the resources is enforced by the firewalls in conjunction with virtual machine service agents. The virtual machine service agents ensure that the agents that they execute only interact with the underlying operating system and remainder of the site's environment through primitives they provide. TACOMA allows agents to be accompanied by digital certificates stored in their xCODE-SIG folder. These certificates are interpreted by service agents as defining accesses permitted by the signed code (Johansen et al., 2002b).

In addition to providing protection for the host, TACOMA also caters, in a limited way, for agent integrity – computations must be protected from faulty or malicious hosts. This is achieved using replication and voting (Johansen et al., 2002a).

6.3.3 WASP (Web Agent-based Service Providing) version 2.0

The WASP project is an initiative of Darmstadt University of Technology with the goal of providing services on web data using mobile agents to implement these services (Fünfrocken, 1997). The WASP system is built upon the idea of using agent technology in conjunction with the WWW- by extending (and not just using) the WWW to provide a ubiquitous mobile agent system. WASP agents rely not only on Java's distributed computing concepts but integrate agent environments into WWW servers with the help of server extension modules to achieve its functionalities. Though WASP operates on a standard Java platform, it supports strong mobility. Other systems such as D'Agents and JAM had to modify the Java Virtual Machine (JVM) in order to capture and restore the Java thread state as is necessary for strong migration. WASP achieves thread level state capture, in a manner similar to that used by Ara for capturing thread state in C language, by using a pre-compiler (Fünfrocken, 1999) and (Cabri et al., 2000a).

In WASP, the EUs called agents are threads in Java interpreter. The interpreter representing the CE forms the environment for the execution of agents called the Server Agent Environment (SAE). The WASP systems achieve their functionality and seamless integrates of SAE into web servers with the help of server extension modules, making use of CGI and Servlets interfaces (Fünfrocken, 1998). In keeping with this, the standard web transport protocol - HTTP is used for agent transfer. WASP uses location dependent URL-style naming convention for naming its agents and SAEs, and resolves them using DNS.

6.3.3.1 Mobility

Being based on Java, WASP implements mobility using Java's object serialization with the potential to migrate to another CE. It makes use of a single primitive to achieve this.

`go()`

This method initiates a HTTP post request to the target destination specified as a parameter to the method. Inside the post request, the agent is transported as Multipurpose Internet Mail Extensions (MIME) message (Fünfrocken et al., 1999). The `go()` primitive offers a proactive migration mechanism enabling strong migration and the bindings in the data space of the migrating EU are removed. The strategy used by WASP for code relocation is lightweight, with only the base classes moved at the time of agents migration and the rest are obtained as the need arises. Both static and dynamic itineraries are supported in WASP. For supporting agent persistence, its internal states are saved on disk for potential later restoration by checkpoint mechanism. This is done at the system level (Fünfrocken, 1998).

6.3.3.2 Communications

Keeping inline with its quest for seamless integration and extension of agent functionalities to the web services, WASP makes use of a modular approach to integrating communication infrastructure to its core architecture. According to Fünfrocken et al. (1999), WASP supports the following mechanisms for agent communication:

6.3.3.2.a Message passing

This method is used for both synchronous and asynchronous communication whether the agents are local or on another SAE. Message passing process entails the exchange of objects between agents.

6.3.3.2.b Stream

Using streams, agents establish a direct connection with a target agent and then send messages across the connection. On migration of the agents the streams are reconnected (if this feature is enabled by the agents on stream creation).

6.3.3.2.c Remote Method Invocation

This is a modular communication mechanism that allows agents to register communication objects that can be connected to by other agents. Any distributed communication mechanisms can be used in this standardized way though currently only CORBA and Java-RMI are supported.

6.3.3.2.d Local Method Invocation

In this mechanism, agents export and import references to Java objects that are accessible at the local SAE.

6.3.3.3 Security

Like most MAS, WASP also provides the basic security mechanisms to protect a host from malicious agents. Data security is provided by means of protection domain called realms. A realm consists of a set of data, specified by local URLs, to which access is restricted. For every realm there is an owner (human or agent) who can define rights for the realm (read, write, execute, etc.) Based on the identities of the agent / user, access to the realm is assigned (Fünfrocken, 1998).

For the purpose of authentication and authorization of agents, WASP makes use of Java Card – a smart card containing Java byte code interpreter. The Java Card in conjunction with SAE is also used as trusted computing base for mobile agents. All such cards have their own private key that is used for decryption of agents and used as a means for authentication of parties involved. Unauthorized access to agent's data during transit is protected by the use of encryption (Fünfrocken, 1999).

6.3.4 Aglets (version 2.0)

Aglets, developed by IBM, is one of the more popular systems developed in Java on Windows platform. The mobile agents are referred to as 'aglets' and migrate between agent servers known as aglet context. The aglet context is the execution environment in which aglets operate. It provides an interface to the underlying operating system through which the aglets are able to access the core facilities and gain reference to other aglets. Aglets make extensive use of event handling methods that can be customized by the programmer to deal with important events in the life cycle of an aglet (Aridor et al., 1998). Aglets, like most Java based systems, implements mobility using Java's object serialization and does not capture thread-level execution state. In other words, aglets supports weak form of mobility.

In Aglets, a EU (called aglet) is a thread in Java interpreter - which constitutes the CE. The aglet context, an abstraction for CE, is the execution environment in which the aglets operate. The aglet context provides an interface to the underlying operating system through which aglets are able to access the core facilities and gain reference to other aglets (Lange et al., 1998). Contexts providing similar services are grouped to form domains. Each aglet runs in its own thread of execution. A distinguishing feature of Aglets is its callback-based programming model. The system invokes certain methods on the agent when certain events in its life cycle occur. Every aglet is assigned a globally unique identity that it keeps throughout its lifetime and is independent of the context in which it executes. The context identity is the URL of the host together with a qualifier if there is more than one context at a host (Kiniry, et al., 1997). The aglet and context names are resolved using DNS services.

6.3.4.1 Mobility

Aglets, like most Java based systems such as Ajanta and Concardia, implement mobility using Java's object serialization and do not capture thread-level execution state. Aglets support weak form of mobility with the potential to push and pull stand-

alone code between the hosts. Aglets provide two primitives to achieve this form of mobility:

`dispatch` (*destination*)

Execution of `dispatch` (*destination*) performs code shipping to a different context specified by its URL in asynchronous and immediate fashion.

`retractAglet` (*destination, agentID*)

Performs code pulling of the agent with identity *agentID* in a synchronous immediate fashion from the destination.

In both cases the aglet is re-executed from a predefined point after migration and retains the values of its objects attribute that are used to provide an initial state for its computation (Lange et al., 1998). The strategy used for code relocation is lightweight, with only the base classes moved at the time of agents migration and the rest are obtained on demand. The attribute values may contain references to resources, which are always managed by copy. Aglets define their own transport protocol – Agent Transfer Protocol (ATP) modeled on top of HTTP for aglet transfers. Agent migration is absolute, as it requires specifying location-dependent URLs for destination servers (using the dispatch primitive). In most of the MAS such as D'Agents, a migration invocation implicitly refers to the current agent; Aglets, however, also support migrating another agent, an operation that can be easily accommodated in the event-based execution model of Aglets. Both static and dynamic itinerary are supported by Aglets architecture. For supporting agent persistence, its internal states are saved on disk for potential later restoration by checkpoint mechanism. Aglets support persistence at both the system and application levels (Karjoth et al., 1997).

6.3.4.2 Communication

Aglets communicate by passing messages between themselves via the context. This process entails the exchange of message objects between aglets, and allows for both

synchronous and asynchronous message passing. Aglets also support “future reply” mechanism where a sender agent of a message does not block until it receives a reply, but rather receives a “future” object and continues with other activities and may poll later to see if the reply has come in. In addition, an aglet can also multicast a message to all aglets within the same context that have subscribed to that message (Karjoth et al., 1997). The primitives used by aglets to achieve the different styles of communication are indicated below.

`sendMessage()` is used to send messages in a synchronous way.

`sendAsyncMessage()` send message in asynchronous way.

`sendOnewayMessage()` sends a one-way message to the aglet. The message is sent asynchronously without the receiver sending a reply.

`sendFutureMessage()` sends a message to the aglet. The message aglet does not block but returns a FutureReply that the sender can poll to see if the message has come in.

`multicastMessage()` is used to multicast a message to all aglets running on the same context.

6.3.4.3 Security

As mentioned earlier, aglets are Java objects and have access to potentially all Java class files on the host; they also relay on the security of the Java interpreter for their proper execution. Thus, aglet security and Java security go hand in hand. The Aglets model features the agent, the host (context), and the host domain, each split into three independent principals as the entity itself, the entity’s owner and its manufacturer. The exception being the host domain as it does not have its manufacturer (in total eight principals). To allow fine-grained control, a security policy consists of a set of named privileges and a mapping from the principals to the privileges are used (Fischmeister et al., 2001).

For aglets transportation, a secure channel is established. The sending context protects the integrity of aglet data by computing a secure hash value that allows the receiving context to perform after-the-fact detection of tampering. Unauthorized access to aglets data during transit is protected by the use of encryption (Fischmeister et al., 2001). Aglets, like Ara implements authentication base on Secure Socket Layer (SSL) protocol. As in most other MAS, Aglets too fall short when considering host attack on its agents (Gong, 1998).

6.3.5 ARA (Agent for Remote Action) version 1.0a

Developed at the University of Kaiserslautern, Ara is a multilanguage Mobile Agent System written in C, C++, Tcl or Java, for portable and secure execution of mobile agents in heterogeneous environment (Peine, 1997). Ara's main design aims are to provide full mobile agent functionality while retaining as much as possible of established programming models and languages- mobility should be integrates as seamlessly as possible with existing programming concepts and avoid remote communication altogether (Peine, 2002). Mobile agents in Ara have the ability to change their host machine during execution while preserving their internal state. This enabled them to handle interactions locally which otherwise would have to be performed remotely. Ara offers full migration of agents, i.e. orthogonal to he conventional program execution, which relieves the programmer of all details involved with remote communication and state transfer.

Ara EU is called agents. Ara systems are similar to D'Agents in a sense that it too has a core service layer supporting multiple languages through interpreters, which in this case forms the CE. The Ara agents move between and stay at places where it uses services provided by the host or other agents (Peine, 1997). The language dependent features, such as state capturing and correctness checking, are the responsibilities of the interpreter, whereas providing access to the underlying operating system services and other mobile agent specific services are the responsibilities of the core. Ara agents

are executed in parallel threads, while some of the internal core functions, for instance the naming service, can be executed as separate processes (Tripathi et al., 2002).

All core objects visible to the programmer such as agents, service points (discussed later in this chapter), and places bear globally unique identities and are used to name the entities when invoking a core function. Places as destinations of migration have names derived from DNS names of their host, optionally extended by a local name to distinguish more than one place on a host. In addition, an Ara place establishes a domain of logically related services under a common security policy governing all agents at that place. For the purpose of name resolution, DNS services are utilized (Peine et al., 1997). Agents may checkpoint their internal states to disk for potential later restoration. Provision for persistence is done at the system level. Ara platform provides for both static and dynamic itinerary. For transportation of agents, Ara supports TCP, SMTP, HTTP, FTP and AX.25 transport protocols (Peine, 1998).

Ara agents make use of path proxies to indicate the path they have followed; by following this path an agent may be located. Unlike D'Agents, which uses an Agent Interface Definition Language to allow agents in different languages to communicate, Ara supports communication is a heterogeneous environment by using different interpreters for each language (Peine et al., 1997).

6.3.5.1 Mobility

Like D'Agents, Ara agents can migrate at any point in their execution carrying along its code, data and execution state, however Ara makes use of a single core call namely `ara_go` to achieve this. In addition to capturing the agents states, the `ara_go` call, also abstracts the complexity of extracting the agent from the local system, marshaling it to another, possibly heterogeneous, machine and reinstalling it there (Peine, 2002).

The syntax of this primitive is as shown below.

`ara_go` *place*

Where *place* is the place name on a particular host to be visited. Ara agents move with all the codes needed at the time of migration. `ara_go` provides a proactive mobility mechanism and the bindings in the data space of the migrating EU are removed. Ara together with Aglets is probably the only MAS that allow one agent to request the migration of another.

6.3.5.2 Communication

Two of Ara's design goals; language independence and avoidance of remote coupling directly influence the choice of its agent communication mechanisms. The latter goal has resulted in avoiding remote communication altogether (Peine, 1997). Ara supports two simple and efficient mechanisms for this - synchronous RPC and tuple space. For message passing, the core uses service points. These mechanisms are discussed below.

6.3.5.2.a Service Points

A service point is a meeting point at a certain place with a unique name where agents collocated at the place can interact in a client-server fashion. An agent creating a service point implicitly becomes its (one and only) server, and any number of agents meeting this service point, become its clients. A client may submit a request (some arbitrary data) to a service point, and the server may fetch that request, process it, and reply with some arbitrary data to this request (Peine et al., 1997). Submitting a request is a synchronous blocked operation, quite analogous to RPC call. Each request is tagged with the name of the client agent, which the server may use when deciding on the reply. The service point implementation handles all cases of unexpected client or server agent migration by terminating gracefully, or arranging for suitable error results to be displayed.

6.3.5.2.b Tuple Space

The tuple space is a means of communication for agents that are, in contrast to service points, not necessarily collocated at the same place at a given moment in time. Ara utilizes the tuple space mechanism for quick access to shared data implemented in main memory. The tuple space is data storage local to a place where shared data is stored in key-value pair (collectively called a tuple). Tuples are indexed by their keys and provide simple operations to read, write, and delete a tuple. In addition to simple read and write operations, a tuple may also be locked in order to synchronize concurrent accesses.

Both mechanisms operate at the level of unstructured byte array data. While this allows easy definition of application dependent interaction, it does offer less programming convenience than e.g. an RPC interface (Peine, 2002).

6.3.5.3 Security

Security in Ara, like in D'Agents, is implemented by executing agents within an interpreter, which controls their resource accesses, and enforces authorization policy expressed in the form of allowances; agents and hosts may be authenticated, agent transfers may be encrypted, and any resource consumption is accounted. In addition to establishing a domain of logically related services under a common security policy, a place's central function is to decide on the condition of admission, if at all, of an agent applying to enter that place. These conditions are expressed in the form of an allowance, a vector of access rights to the resources, approved for the agent for the duration of its stay at that place. It is the function of the Ara core to ensure that an agent never oversteps its allowance (Peine, 1998).

The Ara security model builds in authentication of agents and hosts by using digital signatures and public key certificates. Ara supports authentication base on Secure Socket Layer (SSL) protocol (Peine, 1998). SSL is a connection-oriented, bi-directional point-to-point data transfer protocol offering authentication of both end

points as well as encryption and authentication of the data transferred, offering various cipher suites and certificates (Tan et al., 2002), (Wagner et al., 1996). Like most other MAS, Ara too has as yet not implemented any mechanism for agent protection from malicious hosts.

The next section presents the evaluation of the selected mobile agent systems against the selection factors developed in chapters 3, 4 and 5.

6.4 Evaluation against the selection factors

The evaluations of the above MASs against the criteria are shown in a tabular form below.

6.4.1 Evaluation of the selected MAS against the mobility factors

	D'Agents (Group 1)	TACOMA (Group 2)	WASP (Group 3)	Aglets (Group 4)	ARA (Group 5)
Mobility Model	Strong	Weak	Strong	Weak	Strong
In the case of strong mobility:	Migration and Remote Cloning	N/A	Migration	N/A	Migration
How are the agents mobility achieved					
In the case of strong mobility:	Proactive	N/A	Proactive	N/A	Proactive
Nature of migration / remote cloning					
In the case of weak mobility:	N/A	Push	N/A	Push and Pull	N/A
Direction of code movement					
In the case of weak mobility:	N/A	Stand-alone	N/A	Stand-alone	N/A
Nature of code migrated					
In the case of weak mobility:	N/A	Asynchronous	N/A	Asynchronous and Synchronous	N/A
Synchronisation involved					

In the case of weak mobility: **Time of execution**	N/A	Immediate	N/A	Immediate	N/A
Single or multiple entry points	N/A	Single	N/A	Single	N/A
Strategies used for code relocation	Heavyweight	Preloaded	Lightweight	Lightweight	Heavyweight
Data space management techniques	Binding removal and By copy	By copy	Binding removal	By copy	Binding removal
Nature of agent migration	Absolute only	Absolute and Relative	Absolute only	Absolute only	Absolute only
Agent transportation mechanisms	Standard	Standard	Standard	Standard and proprietary	Standard
Agent transportation protocols	TCP, e-mail	TCP, HTTP, SMTP	HTTP	ATP, HTTP	TCP, SMTP, HTTP, FTP, AX.25
The level at which the MAS supports persistence	None	System level	System level	System level and Application level	System level
Mechanisms used for supporting agent persistence	None	Rearguard, Replication and voting	Checkpoint	Checkpoint	Checkpoint

Nature of the agent's itinerary	Static and Dynamic	Static and Dynamic	Static and Dynamic	Static and Dynamic	Static and Dynamic
Ability of one mobile agent to migrate another	No	No	No	Yes	Yes

Table 8: Evaluation of the selected MAS against the mobility factors

6.4.2 Evaluation of the selected MAS against the communication factors

	D'Agents (Group 1)	TACOMA (Group 2)	WASP (Group 3)	Aglets (Group 4)	ARA (Group 5)
Nature of Message passed	Synchronous & Asynchronous	Synchronous & Asynchronous	Synchronous & Asynchronous	Synchronous & Asynchronous	Synchronous & Asynchronous
Mechanisms to achieve synchronous comm.	Message passing, Streams	Message passing	Message passing, Streams, Local method invocation (LMI)	Message passing	RPC like (via Service point)
Mechanisms to achieve asynchronous comm.	Messages, Events, ARPC	RPC (via exchange of briefcase)	Message passing, CORBA/RMI	Future-reply, message passing	Tuple space
Type of information passed between agents	String messages	String messages	Objects	Objects	Unstructured byte array
Manner in which communicating parties are addressed	Direct	Direct	Direct	Indirect	Direct

Mechanisms for: Agent – Agent Comm.	Intra-place	Inter-place	Intra-place	Inter-place	Intra-place	Inter-place	Intra-place	Inter-place	Intra-place	Inter-place
	Message passing, streams, Events, ARPC	Message passing, streams, Events, ARPC	Message passing	Message passing	Message passing, Streams, LMI	Message passing, Streams	Message passing	Message passing	RPC	Not permitted
Mechanisms for: Agent – Service agent comm.	Intra-place	Inter-place	Intra-place	Inter-place	Intra-place	Inter-place	Intra-place	Inter-place	Intra-place	Inter-place
	ARPC	Message passing	RPC	Message passing	Message passing, RMI	Message passing	RMI	Message passing	Message passing	Not Allowed
Mechanisms for: Agent – Agent Group comm.	Intra-place	Inter-place	Intra-place	Inter-place	Intra-place	Inter-place	Intra-place	Inter-place	Intra-place	Inter-place
	Tuple Space	Not Supported	Wrappers	Not Supported	Not supported	Not Supported	Event based	Not Supported	Tuple Space	Not Supported
Naming mechanisms for agents	Global Unique ID Based on host name		Global Unique ID Based on Host name		Global Unique ID Based on Host name		Global Unique ID independent of host name		Global Unique ID Based on Host name	
Names constant throughout agent life-cycle	No		No		No		Yes		No	
Names dependant on location of execution	Yes		Yes		Yes		No		Yes	

Support location transparency at application layer?	Yes	Yes	Yes	Yes	Yes
Mechanisms for locating agents	Logging: Database (decentralized) & path proxies (navigational agents)	Path proxies (via Rearguard)	Logging: Database (decentralized)	Logging: Database (decentralized), path proxies, Advertising	Logging: Path proxy
Naming Mechanisms for place	DNS based names	DNS based names	Host URL + other quantifier	Host URL + other quantifier (if more than one places on a host)	DNS based names
Name resolution mechanisms	DNS	DNS	DNS	DNS	DNS
Interoperability standard supported	None	None	None	MASIF	None
Compliant to ACL	Not supported	Not supported	Not supported	Not supported	Not supported
Supports comm. in heterogeneous language?	Yes	Yes	No	No	Yes
Mechanisms to support communication in heterogeneous language	AIDL	Interpreters for the supported language	Java Only	Java Only	Interpreters for the supported language

Table 9: Evaluation of the selected MAS against the communication factors

6.4.3 Evaluation of the selected MAS against the security factors

	D'Agents (Group 1)	TACOMA (Group 2)	WASP (Group 3)	Aglets (Group 4)	ARA (Group 5)
Mechanisms supported for agent confidentiality	Encryption using PGP	Encryption (algorithm could not be determined)	Using hardware (Java Card)	Encryption using SSL	Encryption using SSL
Mechanisms to support agent integrity	Digital signature using PGP	Digital signature	Using hardware (Java Card)	Digital signature using SSL	Digital signature using SSL
Mechanisms for authentication of the agent's owner	Digital signature using PGP	Digital signature	Using Hardware (encryption algorithm could not be determined)	Digital signature using SSL	Digital signature using SSL
Mechanisms used for authorization and access control	Resource Management agent via access right list	Virtual machine service agent via access right list	SAE via Access right list	Context via policies	Places via allowances
Mechanisms to protect host: From attack by the agents located at the host	Padded cells	Padded cells	Sandbox	Sandbox	Padded cells
Mechanisms to protect host: From attack by other hosts in the system	Encryption of messages or agents before transmission	Firewall & Encryption of messages or agents before transmission	Encryption of messages or agents before transmission	Encryption of messages or agents before transmission	Encryption of messages or agents before transmission
Mechanisms to protect agents: From attack by the host	None	None	Using Hardware (Java Card)	None	None

Mechanisms to protect agents: From attack by other agents co-located at the host	Padded cells	Padded cells	Sandbox	Sandbox	Padded cells

Table 10: Evaluation of the selected MAS against the security factors

7 Conclusions

7.1 Introduction

The main objective of this dissertation was to establish means that could be used for selecting mobile agent systems. This was necessitated by the fact that there are many mobile agent systems performing similar functions. To achieve this objective, the analysis of the fundamental requirements of mobile agent systems, which were identified (in section 1.3 of chapter 1) to be mobility, communication and security, was performed. Based on this analysis, factors for evaluating mobile agent systems were proposed (sections 3.9, 4.6 and 5.6 of chapters 3, 4 and 5 respectively). These factors were then used to evaluate mobile agent systems (section 6.4 of chapter 6).

As it was not feasible to evaluate all mobile agent systems, mobile agent systems with common properties were grouped together (in other words, if a statement is true for one agent system of the group then it is probably true for all members of that group). An agent system from each of the groups was selected and evaluated. This evaluation provides a means to compare various agent systems. From this comparison, presented in section 6.4 of chapter 6, selection of mobile agent systems with desired properties can be made. Although this dissertation evaluates only five mobile agent systems, the evaluation holds true for all other agent systems belonging to those groups as each selected agent system belonged to a group with similar fundamental properties.

7.2 Interpretation of Results

This section presents some deductions on the findings presented in the previous chapter (chapter 6).

With respect to agent mobility, the following deductions can be made:

- In agent systems that support strong mobility (groups 1, 3 and 5), both migration and remote cloning is used as mechanisms for achieving mobility in agents and the nature of migration is predominantly proactive. In these systems, strategies supported for code relocation is heavyweight except in cases of agent systems developed using Java. In agent systems supporting weak form of mobility (groups 2, and 4), code is normally pushed to the remote host in an asynchronous manner although some systems may support synchronous and pull code mechanism as well (section 6.4.1).

- The data space management techniques used for agents systems supporting strong mobility is largely by binding removal whereas in systems supporting weak forms of mobility it is chiefly by copy. In all groups of agent systems, the nature of agent's itinerary support is both static and dynamic, the nature of agent migration is predominantly absolute although some agent systems also support relative migration. In addition most of the agent systems make use of standard agent transportation mechanism (section 6.4.1).

- Agent systems that support persistence, implement it mainly at system level and use checkpoint as the chief mechanism (section 6.4.1).

From the information presented in this study, the following conclusions can be drawn with regards to issues related to agent communication:

- All the groups support communication in both synchronous and asynchronous manner and the predominant mechanism to achieve this is message passing. In majority of the agent systems, the types of information passed between agents

are either just strings (groups 1 & 2) or are in the form of objects (groups 3 & 4) and the communicating parties address each other directly (section 6.4.2).

- Of the many mechanisms to support agent-to-agent and agent-to-service agent (intra-place and inter-place) communication, the most common one is message passing. When considering mechanisms for agent-to-group (intra-place) communication, a common approach adopted is tuple space. It must be noted that not all agent systems provide mechanisms for this (intra-place) type of communication and none of the agent systems support communication between agents and groups of agents when they are situated on different places (section 6.4.2).

- For agent naming purposes, most of the agent systems use globally unique ID that is based on host names. Because these names are based on current host the agent is at, they are not constant throughout the agent's life. In spite of the fact that the agent names are based on its location, all groups of agent systems provide for location transparency at application layer. The naming of places is based mainly on DNS names and the host URLs. In all cases they are resolved by making use of domain name services (section 6.4.2).

- With respect to interoperability, none of the groups are compliant to an agent communication language and most do not support any interoperability standards. Agent systems belonging to groups 1, 2 and 5 support communications in heterogeneous languages by making use of interpreters for the various languages or using agent interface definition languages. Groups 2 and 4 support communications in a homogeneous language (Java)(section 6.4.2).

From the analysis of agent systems security, the following deductions can be made:

- Majority of the agent systems support agent confidentiality by making use of encryption algorithms. Agent integrity and authentication is provided by making use of digital signatures and digital certificates. In most agent systems,

the authorizations to access resources are managed either by special agents or the places themselves and access control lists provide control to resources (section 6.4.3).

- Majority of agent systems use padded cells or sandbox technique to protect the host from attack by agents located at the host. These techniques are also used to protect agents from attack by other agents co-located at the host. A host protects itself from attack by other hosts in the system, predominantly by encryption of messages or agents before transmission. The only way available to agents to protect itself from attack by the host is by making use of hardware (section 6.4.3).

7.3 Further Research

For the selection of mobile agent systems, this study focused on the implementation aspects of the systems, in particular the issues surrounding mobility, communication and security. For this reason the study did not include the following aspects. A possible extension pertaining to this study could include:

- A study could be made to select agent systems based on their performances such as time taken to achieve certain tasks.

- Selection of mobile agent systems could also include the operating systems supported and the nature of support provided by the operating systems for the core functions of the agent systems.

- Selection criteria based on the design, implementation and testing of agents and agent platforms could also be used.

In addition to the extensions suggested above, agent selection criteria based on the structure and other related issues of the agent / execution environment programming languages could also be used.

Appendix A

A.1 Introduction

This appendix presents the source code of the application presented in chapter 2. To recap, this application was developed to show how agents move from one host to another and achieve mobility and how they communication with one another. This application makes use of three agents one of which is a stationary agent and the other two are mobile agents. Once created, the mobile agents move to two different hosts and then initiate their communication with each other. Any communication from the stationary agent is sent simultaneously to both the mobile agents. Each of the mobile agents can communicate independently with the stationary agent. The stationary agent communicates with the two mobile agents in a synchronous manner whereas the two mobile agents communicate with each other asynchronously. As a means to demonstrate basic security, the application allows for the creation of agents only on one of the hosts - the home base.

It is to be noted that the application has been developed using the Aglet Software Development Toolkit (ASDK) version 2.0.1.

A.2 The Source Code

```
// ------------------------------------------------------------------
// The class responsible for creating the GUI
// for the three aglets.
//-------------------------------------------------------------------


package my.aglets;

import com.ibm.aglet.util.AddressChooser;
import javax.swing.*;
import java.awt.event.*;
import java.util.*;
import java.awt.*;

public class GroupWindow extends JFrame implements ActionListener{
      JTextArea textArea;
      JTextField inputField;
      AddressChooser dest;
      String address;
      JPanel panel;
      GroupCMaster master;
      GroupCChild1 child1;
      GroupCChild2 child2;

//-------------------------------------------------------------------
// The constructor for the master frame.
// initilizes the global variales,
// creates and adds the components and the action listener.
//-------------------------------------------------------------------

      public GroupWindow(GroupCMaster cm){
            setTitle("Master Frame");
            setSize(300,300);
            address ="";
            dest = null;
            master=null;
            child1=null;
            child2=null;
            master = cm;

            textArea= new JTextArea();
            inputField= new JTextField();
            panel= new JPanel();
            dest = new AddressChooser();
            panel.setLayout(new BorderLayout());
            panel.add(textArea, "Center");
            panel.add(inputField,"South");
            panel.add(dest, "North");

            getContentPane().add(panel);
            textArea.setEditable(false);
            inputField.addActionListener(this);
            addWindowListener(new WindowAdapter(){
               public void windowClosing(WindowEvent we){
                  setVisible(false);
                  }
            } );
        }
```

```
//---------------------------------------------------------------
// The constructor for the first child frame.
// initilizes the global variales,
// creates and adds the components and the action listener.
//---------------------------------------------------------------

      public GroupWindow(GroupCChild1 cd){
            setTitle("Child Frame");
            setSize(300,300);
            textArea= new JTextArea();
            inputField= new JTextField();
            address ="";
            dest = null;
            master = null;
            child1=null;
            child2=null;
            child1 = cd;
            panel= new JPanel();
            panel.setLayout(new BorderLayout());

            panel.add(textArea, "Center");
            panel.add(inputField,"South");
            getContentPane().add(panel);

            textArea.setEditable(false);
            inputField.addActionListener(this);

      }

//---------------------------------------------------------------
// The constructor for the second child frame.
// initilizes the global variales,
// creates and adds the components and the action listener.
//---------------------------------------------------------------

      public GroupWindow(GroupCChild2 cd2){
            setTitle("Child2 Frame");
            setSize(300,300);
            textArea= new JTextArea();
            inputField= new JTextField();
            address ="";
            dest = null;
            master = null;
            child1=null;
            child2 = cd2;
            panel= new JPanel();
            panel.setLayout(new BorderLayout());

            panel.add(textArea, "Center");
            panel.add(inputField,"South");
            getContentPane().add(panel);

            textArea.setEditable(false);
            inputField.addActionListener(this);

      }
```

```
//-----------------------------------------------------------------
// This method deals with the events generated by the text field.
//-----------------------------------------------------------------

      public void actionPerformed(ActionEvent ae){
            Object source = ae.getSource();
            if(source==inputField){
               String s = inputField.getText();
               textArea.append(s+ "\r\n");
               if(master !=null){
                   if(!address.equals(dest.getAddress()))
                     master.dispatchChild(address=dest.getAddress());
                     master.sendText(s);
                }

                if(child1 != null){
                  child1.sendText(s);
                }
                  if(child2 != null){
                    child2.sendText(s);
                  }
                inputField.setText("");
             }
       }

//-------------------------------------------------------------------
// adds the information to the TextArea from the TextField
//-------------------------------------------------------------------

       public void appendText(String s){
          textArea.append(s+ "\r\n");
       }
}
```

```
//-------------------------------------------------------------------
//The GroupCmaster aglet class responsible for creating
// the two children aglets. The master is a stationary
// agent and the two children are mobile agents.
// On creation, the mobile agents are dispatched to
// two different host.
//-------------------------------------------------------------------

package my.aglets;

import com.ibm.aglet.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.net.*;

public class GroupCMaster extends Aglet{
      transient AgletProxy remoteProxy;
      transient AgletProxy remoteProxy2;
      String name;
      GroupWindow window;
      AgletProxy chProxy;
      Message msg3;

//-------------------------------------------------------------------
// The constructor of the stationary agent
// responsible for initialization of the global variables.
//-------------------------------------------------------------------

      public GroupCMaster(){
            remoteProxy = null;
            remoteProxy2 = null;
            window = null;
            name ="Unknown";
            msg3=null;

      }


//-------------------------------------------------------------------
// The  method responsible for creating and dispatching
// the mobile agents to other hosts in the system.
//-------------------------------------------------------------------

      public void dispatchChild(String s){
            try{
                URL url= new URL(s);
                if(remoteProxy!=null && remoteProxy2 != null){
                  remoteProxy.sendMessage(new Message("bye"));
                  remoteProxy2.sendMessage(new Message("bye"));
                }
                if(s.equals("atp://venus:4434")){
                  AgletContext ac= getAgletContext();
                  AgletProxy ap = ac.createAglet(null,
                  "my.aglets.GroupCChild1",getProxy());
                  AgletContext ac2= getAgletContext();
                  AgletProxy ap2 = ac2.createAglet(null,
                  "my.aglets.GroupCChild2",getProxy());
                  remoteProxy = ap.dispatch(url);
                  remoteProxy2 = ap2.dispatch(new
                  URL("atp://saturn:4434/"));
```

```
                }
                  else
                    if(s.equals("atp://saturn:4434")){
                      AgletContext ac= getAgletContext();
                      AgletProxy ap = ac.createAglet(null,
                      "my.aglets.GroupCChild1",getProxy());
                        AgletContext ac2= getAgletContext();
                        AgletProxy ap2 = ac2.createAglet(null,
                        "my.aglets.GroupCChild2",getProxy());

                        remoteProxy = ap.dispatch(url);
                        remoteProxy2 = ap2.dispatch(new
                        URL("atp://venus:4434/"));
                     }

              } catch(InvalidAgletException iae){
                  iae.printStackTrace();
            } catch(Exception e){
                  e.printStackTrace();
            }
       }


//-----------------------------------------------------------------------
// The method responsible for getting the system property.
//-----------------------------------------------------------------------

      private String getProperty(String s){
            return System.getProperty(s, "Unknown");
      }


//-----------------------------------------------------------------------
// The method responsible for handling messages sent
// to the stationary agent.
//-----------------------------------------------------------------------
      public boolean handleMessage(Message message){
            if(message.sameKind("dialog"))
               window.show();
             else
                if(message.sameKind("text")){
                        if(!window.isVisible())
                              window.show();

                  window.appendText((String)message.getArg());
              return true;
            }
               if(message.sameKind("msg")){
                  try{
                        AgletProxy chProxy =
                        (AgletProxy)message.getArg();
                         chProxy.toString();
                        remoteProxy2.sendMessage(new
                        Message("msg3",chProxy));
                        }catch(Exception e){
                           e.printStackTrace();
                           }
                   return true;
                }
         return false;
       }
```

```
//-----------------------------------------------------------------------
// This method creates and shows the GUI for the stationary
// agent upon the agents creation.
//-----------------------------------------------------------------------

      public void onCreation(Object obj){
            window = new GroupWindow(this);
            window.show();
            try{
                  name= getProperty("user.name");
             } catch(Exception e){
                   e.printStackTrace();

               }
         }


//-----------------------------------------------------------------------
// This method is called when the agent is being disposed.
// It disposes off the GUI  and sends a message to the mobile
// agents at the remote hosts.
//-----------------------------------------------------------------------

      public void onDisposing(){
            if(window != null){
                  window.dispose();
                  window=null;
            }
            if(remoteProxy != null){
                  try{
                        remoteProxy.sendMessage(new Message("bye"));
                  } catch(AgletException ae){
                        ae.printStackTrace();
                     }
            }
            if(remoteProxy2 != null){
                  try{
                        remoteProxy2.sendMessage(new Message("bye"));
                  } catch(AgletException ae){
                        ae.printStackTrace();
                     }
            }
      }

//-----------------------------------------------------------------------
// This method sends messages to the mobile agents at remote hosts
//-----------------------------------------------------------------------

      public void sendText(String s){
            try{
               if(remoteProxy != null)
                  remoteProxy.sendMessage(new
                  Message("text",name+":"+s));
                  remoteProxy2.sendMessage(new
                  Message("text",name+":"+s));
             } catch(Exception e){
                   e.printStackTrace();
               }
            }
      }
```

```
//-----------------------------------------------------------------------

// This class is responsible for handling communication
// between the first mobile agent (child1) and the other agents in
// the system.
//-----------------------------------------------------------------------

package my.aglets;

import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.awt.event.*;

public class GroupCChild1 extends Aglet{

      transient String name;
      transient GroupWindow window;
      AgletProxy masterProxy,apc1;
      String sapc1;
      Message msg;


//-----------------------------------------------------------------------
// constructor to initialize the global variables
//-----------------------------------------------------------------------

      public GroupCChild1(){
            name="Unknown";
            window= null;
            masterProxy = null;
            sapc1="";
            msg=null;
      }


//-----------------------------------------------------------------------
// The method responsible for getting the system property.
//-----------------------------------------------------------------------

      private String getProperty(String s){
            return System.getProperty(s, "Unknown");
      }

//-----------------------------------------------------------------------
// The method responsible for handling communication sent
// to it and establishing a link between the other mobile agent
// and itself for asynchronous communication.
//-----------------------------------------------------------------------

      public boolean handleMessage(Message message){

            if(message.sameKind("dialog")){
                  window.show();
            } else {
                  if(message.sameKind("text")) {
                        String s = (String)message.getArg();
                        if(!window.isVisible())
```

```
                        window.show();
                  window.appendText(s);
                  return true;
                  }
                  if(message.sameKind("msg")){
                     try{
                        AgletProxy ch1Proxy =
                        (AgletProxy)message.getArg();
                        masterProxy.sendMessage("msg",ch1Proxy);
                        future= ch1Proxy.sendFutureMessage(new
                        Message("text",name+":"+s));
                           if(future.isAvailable()){
                              String reply = (String)
                              future.getReply();
                              window.appendText(reply);
                           }

                         }catch(Exception e){
                              e.printStackTrace();
                          }
                     return true;
                   }

                if(message.sameKind("bye")){
                   window.appendText("Bye Bye...");
                      try{
                         Thread.currentThread();
                         Thread.sleep(3000L);
                      } catch(Exception e){
                              e.printStackTrace();
                          }
                      message.sendReply();
                      dispose();
                  }
            }

            return false;
      }


//---------------------------------------------------------------------
// On creation of the agent, this method adds the
// mobility capabilities to the agent and displays the
// GUI.
//---------------------------------------------------------------------

      public void onCreation(Object obj){
            masterProxy = (AgletProxy)obj;
            AgletProxy apc1 = getProxy();
            try{
                  masterProxy.sendMessage(new Message("msg",apc1));
            }catch(Exception e){
                  e.printStackTrace();
               }

            addMobilityListener(new MobilityAdapter(){
               public void onArrival(MobilityEvent me){
                  try{
                     window = new GroupWindow(GroupCChild1.this);
                     window.show();
                     name=getProperty("user.name");
```

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



//---------------------------------------------------------------------
// This method is called when the agent is being disposed.
// It disposes off the GUI.
//---------------------------------------------------------------------

      public void onDisposing(){
            if(window != null){
                  window.dispose();
                  window=null;
            }
      }

//---------------------------------------------------------------------
// This method sends messages to both the stationary and
// the mobile agent.
//---------------------------------------------------------------------

      public void sendText(String s){
         try{
           if(masterProxy != null)
             masterProxy.sendMessage(new Message("text",name+":"+s));
             masterProxy.sendMessage(new Message("msg",apc1));
            } catch(Exception e){
                  e.printStackTrace();
              }
          }
      }
```

```
//-----------------------------------------------------------------------

// This class is responsible for handling communication
// between the second mobile agent (child2)and the other agents in
// the system.
//-----------------------------------------------------------------------

package my.aglets;

import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;

public class GroupCChild2 extends Aglet{

      transient String name;
      transient GroupWindow window;
      AgletProxy masterProxy,ch1Proxy;
      FutureReply future;


//-----------------------------------------------------------------------
// constructor to initialize the global variables
//-----------------------------------------------------------------------
      public GroupCChild2(){
            name="Unknown";
            window= null;
            masterProxy = null;
            ch1Proxy = null;
            future = null;
      }


//-----------------------------------------------------------------------
// The method responsible for getting the system property.
//-----------------------------------------------------------------------

      private String getProperty(String s){
            return System.getProperty(s, "Unknown");
      }


//-----------------------------------------------------------------------
// The method responsible for handling communication sent
// to it and establishing a link between the other mobile agent
// and itself for asynchronous communication.
//-----------------------------------------------------------------------

      public boolean handleMessage(Message message){
            if(message.sameKind("dialog")){
                  window.show();
            } else {
                  if(message.sameKind("text")) {
                        String s = (String)message.getArg();
                        if(!window.isVisible())
                            window.show();
                    window.appendText(s);
                    AgletProxy ch1Proxy =
                    (AgletProxy)message.getArg();
```

```
                return true;
                }


                if(message.sameKind("msg3")){
                   try{
                      AgletProxy ch1Proxy =
                      (AgletProxy)message.getArg();

                masterProxy.sendMessage("msg",ch1Proxy);
                future= ch1Proxy.sendFutureMessage(new
                Message("text",name+":"+s));
                   if(future.isAvailable()){
                      String reply = (String) future.getReply();
                            window.appendText(reply);
                       }

                 }catch(Exception e){
                      e.printStackTrace();
                     }
             return true;
          }

                if(message.sameKind("bye")){
                      window.appendText("Bye Bye...");
                      try{
                            Thread.currentThread();
                            Thread.sleep(3000L);

                      } catch(Exception e){
                            e.printStackTrace();
                        }
                      message.sendReply();
                      dispose();
                }
          }
          return false;
       }


//---------------------------------------------------------------------
// On creation of the agent, this method adds the
// mobility capabilities to the agent and displays the
// GUI.
//---------------------------------------------------------------------

      public void onCreation(Object obj){
            masterProxy = (AgletProxy)obj;
            addMobilityListener(new MobilityAdapter(){
                  public void onArrival(MobilityEvent me){
                        window = new GroupWindow(GroupCChild2.this);
                        window.show();
                        try{
                              name=getProperty("user.name");

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

```
//-----------------------------------------------------------------------
// This method is called when the agent is being disposed.
// It disposes off the GUI.
//-----------------------------------------------------------------------

      public void onDisposing(){
            if(window != null){
                  window.dispose();
                  window=null;
            }
      }


//-----------------------------------------------------------------------
// This method sends messages to both the stationary and
// the mobile agent.
//-----------------------------------------------------------------------

      public void sendText(String s){
            try{
               if(masterProxy != null){
                  masterProxy.sendMessage(new
                  Message("text",name+":"+s));
                  ch1Proxy.sendFutureMessage(new
                  Message("text",name+":"+s));
                }
              } catch(Exception e){
                  e.printStackTrace();
                }
       }

}
```

References

ALTMANN, J., GRUBER, F., KLUG, L., STOCKNER, W. & WEIPPL E. 2000. Evaluation of Agent Platforms, Technical Report 0064/2000, Software Competence Center, Hagenberg. Available at http://www.scch.at/index.jsp [Accessed on 12/2/04].

APPEL, A. W., AND MICHAEL, N. G. 2000. Machine Instruction Syntax and Semantics in Higher Order Logic. In *Proceedings of 17th International Conference on Automated Deduction (CADE-17)*, Lecture Notes in Artificial Intelligence. Springer-Verlag.

ARIDOR, Y., AND OSHIMA, M. 1998. Infrastructure for Mobile Agents: Requirements and Design, In *Proceedings of the Second International Workshop on Mobile Agents (MA'98)*, LNCS, Vol. 1477, pp. 38-49, Stuttgart, Germany.

ARNOLD, K., O'SULLIVAN. B., SCHEIFFLER, R.W., WALDO, J., AND WOLLRATH, A. 1999. *The Jini Specification*, Addison-Wesley, Reading, Mass.

ASSIS, S. F. M., POPESCU-ZELETIN, R. 1998. An approach for providing mobile agent fault tolerance, In *Proceedings of the 2nd International Workshop on Mobile Agents (MA'98)*, LNCS, Vol. 1477, pp. 14-25, Stuttgart, Germany.

BERNA-KOES, M., NOURBAKHSH, I., AND SYCARA, K. 2004. Communication Efficiency in Multi-agent Systems, In *Proceedings of ICRA 2004*. New Orleans, LA.

BETTINI, L., FERRARI, G., PUGLISE, R. 2002. Global programming and mobile code, Technical report 05/2002. Available at http://music.dsi.unifi.it/papers.html, [Accesses on 13/1/2005]

BAUMANN, J. 1999. A Comparison of Mechanisms for Locating Mobile Agents. Available at ftp://ftp.informatik.uni-stuttgart.de/pub/library/ ncstrl.ustuttgart_fi /TR-1999-11/TR-1999-11.pdf. [Accessed on 2/3/2004].

BAUMANN, J., AND ROTHERMEL, K. 1998. The Shadow Approach: An Orphan Detection Protocol for Mobile Agents, In *Proceedings of the 2nd Int. Workshop on Mobile Agents (MA'98)*, LNCS, Vol. 1477, pp. 2-13, Springer-Verlag, Berlin.

BAUMANN, J., HOHL, F., RADOUNIKIS, N., ROTHERMEL K., STRAβER, M. 1997. Communication Concepts for Mobile Agent Systems, In *Proceedings of the First International Workshop on Mobile Agents (MA'97),* LNCS, Vol. 1219, pp. 16-25, Springer-Verlag, Berlin.

BIERMANN, E., CLOETE, E. 2002. Classification of Malicious Host Threats in Mobile Agent Computing, In *Proceedings of SAICSIT,* pp. 141-148.

BLAKLEY, B. 1996. The Emperor's Old Armor. In *Proceeding of the New Security Paradigms Workshop*, pp. 2-16, ACM Press.

BOGGS, J.K. 1973. IBM Remote Job Entry Facility: Generalised Subsystem Remote Job Entry Facility, *IBM Technical Disclosure Bulletin*, 752.

BORSELIUS, B. 2002. Mobile Agent Security, *Electronics and Communication Engineering Journal*, Vol. 14, No. 5, pp 211-218.

BRADSHAW J M., GREAVES M, HOLMBACK H, KARYGIANNIS T, SILVERMAN B, SURI N AND WONG A. 1999. Agents for the masses? *IEEE Intelligent System*, Vol. 14, No. 2, pp. 53-63.

BRAZIER, F. M. T., OVEREINDER, B. J., VAN STEEN, M., AND WIJNGAARDS, N. J. E. 2002. Agent Factory: Generative Migration of Mobile Agents in Heterogeneous Environments, In *Proceedings of the 17th ACM Symposium on Applied Computing*, pp. 101 –106, Spain.

CABRI, G., LEANARDI, L., ZAMBONELLI, F. 2001. Coordination infrastructures for mobile agents, *Microprocessor and Microsystems*, Vol. 25, pp. 85-92.

CABRI, G., LEANARDI, L., ZAMBONELLI, F. 2000a. Mobile-agent coordination models for Internet applications, *IEEE Computer*, Vol. 33, No. 2, pp. 82-89.

CABRI, G., LEONARDI, L., ZAMBONELLI, F. 2000b. Weak and Strong Mobility in Mobile Agent Applications, In *Proceedings of the 2nd International Conference and Exhibition on The Practical Application of Java*, Manchester, U.K. Available at http://polaris.ing.unimo.it/MOON/papers/ pdf/pajava00.pdf [Accessed on 2/3/04].

CHAKRAVARTI, A. J., WANG, X., JASON O. HALLSTROM, J. O., BAUMGARTNER, G. 2003. Implementation of Strong Mobility for Multi-Threaded Agents in Java, *IEEE International Conference on Parallel Processing*, pp. 321

CHAN, P. C., WEI, V. K. 2002. Preemptive Distributed Intrusion Detection Using Mobile Agents, *Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'02*), pp103-108.

CHEN, W. S. E AND LENG, C. W. R. 1997. A Novel Mobile Agent Search Algorithm, In *Proceedings of the First International Workshop on Mobile Agents (MA'97)*, LNCS, Vol. 1219, pp. 162-173, Springer-Verlag, Berlin.

CHUNG F., CHYI N. C. 2003. A Sliding-agent-group Communication Model for Constructing a Robust Roaming Environment Over Internet, *Mobile Network and Applications*, Vol. 8, No. 1, pp. 61-74

CLAESSENS, J., PRENEEL, B., VANDEWALLE, J. 2003. (How) can mobile agents do secure electronic transactions on untrusted hosts? A survey of the security issues and the current solutions, *ACM Transactions on Internet Technology (TOIT)* Vol. 3, No. 1 pp. 28 – 48.

CORRADI, A., MONTANARI, R., AND STEFANNELLI, C. 2001. Security of mobile agents on the Internet, *Internet Research: Electronic Networking Applications and Policy*, Vol. 11, No. 1, pp. 84-95.

CUGOLA, G., DI NITTO, E., FUGGETTA, A. 1998. Exploring an Event-based Infrastructure to Develop Complex Distributed Systems, In *Proceeding of the 20th International Conference on Software Engineering*, pp. 261-270, Kyoto, Japan.

CUGOLA, G., GHEZZI, C., PICCO, G.P., AND VIGNA G. 1997. Analyzing Mobile Code Languages. J. Vitek and C Tschudin, (Eds.), *Mobile object Systems: Towards the programmable Internet*, LNCS, Vol. 1222, pp. 123-132, Springer. Germany.

DIKAIAKOS, M. D., SAMARAS, G. 2000, Workshop on infrastructure for scalable Multi-Agent systems, 4th international conference on Autonomous agents, Available at http://www.cs.ucy.ac.cy/mdd/talks/agents2000.pdf [Accessed on 22/1/2005].

FANG, Q., GAO, J., GUIBAS, L. J. 2004. Locating and bypassing Routing Holes in Sensor Networks, *IEEE Infocom*, Avaliable at http://www.ieee-infocom.org/ 2004/Papers/51_3.PDF [Accessed on 20/5/05]

FARMER, W.M., GUTTMAN, J.D., AND SWARUP, V. 1996. Security for mobile agents: Issues and requirements, *Nat'l. Info.Sys. Security Conf., NISSC*. Available at http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper033/ SWARUP96.pdf

FAROOK, A. H., HIROKI, S. 2003. Dynamic Information Allocation through Mobile Agents to Achieve Load Balancing in Evolving Environment, In *IEEE proceedings of the 6th International Symposium on Autonomous Decentralized Systems (ISADS'03)*, pp 25-33.

FIKES, R., AND FARQUHAR, A. 1999. Distributed Repositories of Highly Expressive Reusable Ontologies, *IEEE Intelligent Systems*, Vol.14, No. 2, pp. 73-79.

FININ, T., LABROU, Y., AND PENG, Y. 1998. Mobile Agents can Benefit from Standards Efforts on Interagent Communication, *IEEE Communication Magazine*, Vol. 36, No. 7, pp. 50-56.

FISCHMEISTER, S., VIGNA G., KEMMERER, R. A. 2001. Evaluating the Security of Three Java-Based Mobile Agent Systems, In G. P. Picco (Ed.), LNCS 2240, pp. 31-44.

FUGGETTA A.,PICCO G. P., VIGNA G. 1998. Understanding Code Mobility, *IEEE transactions on software Engineering*, Vol. 24, No. 5, pp. 342-361.

FÜNFROCKEN, S. AND MATTERN, F. 1999. Mobile Agents as an Architectural Concept for Internet-based Distributed Applications - The WASP Project Approach, In: *Steinmetz (ed.) Proc. Kommunikation in Verteilten Systemen (KiVS'99)*, pp. 32-43, Springer-Verlag. Germany.

FÜNFROCKEN, S. 1997. How to Integrate Mobile Agents into Web Servers, *In Proceedings of 6th IEEE Workshop in Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 94-99, Boston, USA.

FÜNFROCKEN, S. 1999. Protecting Mobile Web Commerce Agents with Smartcards, In *Proceedings of the First International Symposium on Agent Systems and Applications and the Third International Symposium on Mobile Agent Systems (ASA/MA'99)*, pp. 90-102.

FÜNFROCKEN, S. 1998. Transparent Migration of Java-based Mobile Agents (Capturing and Reestablishing the State of Java Programs), In: *Kurt Rothermel, Fritz Hohl (Eds.), Proceedings of Second International Workshop on Mobile Agents (MA'98)*, LNCS 1477, pp. 26-37, Springer-Verlag, Berlin, Germany.

GENESERETH, M. R. & KETCHPEL, S.P. 1994. Software Agents. *Communication of the ACM.* Vol. 37 No. 7, pp. 48-53.

GONG, L. 1998. Secure Java Class Loading. *IEEE Internet Computing,* Vol. 2, No. 6, pp. 56-61.

GONG, L. 1997. Java Security: present and near future. *IEEE Micro,* Vol. 17, No. 3, pp. 14-19.

GRAY, R S, CYBENKO, G., KOTZ, D., PETERSON, R.A., RUS, D. 2002. D'Agents: Applications and performance of a mobile agent system. *Software: Practice and Experience*, Vol. 35, No. 6, pp. 534-573.

GRAY, R. S., KOTZ, D., CYBENKO, G., RUS, D. 1998. D'Agents: Security in a multiple-language, mobile-agent system. In Giovanni Vigna Ed., *Mobile Agents and Security*, LNCS, Vol. 1419, pp. 154-187, Springer-Verlag.

GRAY, R. S., KOTZ, D., CYBENKO, G., RUS, D. 2001. Mobile agents: Motivations and state-of-the-art systems, In Jeffrey Bradshaw (Ed). *Handbook of Agent Technology*, AAAI/MIT press.

GRAY, R.S. 1996. Agent Tcl: A flexible and secure mobile-agent system, In *Proceedings of the Fourth Annual Tcl/Tk Workshop (TCL 96)*, pp 9-23.

GRAY, R.S. 1995. Agent Tcl: A transportable agent system, In *Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95)*, Baltimore, Maryland. Available at http://agent.cd.Dartmouth.edu/papers /gray:agenttcl.pdf [Accessed on 23/2/04].

GREENBERG, M.S., BYINGTON, J.C., HOLDING, T., HARPER, D.G. 1998. Mobile agents and Security, *IEEE Communication Magazine*, Vol. 36, Issue 7, pp. 76-85.

GROSOF, B.N., LABROU, Y. 1999. An approach to using XML and a Rule-based Content Language with an Agent Communication Language, In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) workshop on Agent*

Communication Languages, Stockholm. Germany. Available at www.research.ibm.com/rules/paps/rc21491.ps [Accessed on 1/5/04].

GSCHWIND T., FERIDUN, M., AND PLEISCH S. 1999. ADK- Building Mobile Agents for Network and Systems management from Reusable Components. In *Proceedings of the First International Symposium in Agent Systems and Applications and Third International Symposium on Mobile Agents.*

GSCHWIND, T., RMI64, 1999 http://www.infosys.tuwien.ac.at/Staff/Tom/Projects/rmi64, [Accessed on 12 August 2004].

HARRISON, C.G., CHESS, D. M., KERSHENBAUM, A.1995. Mobile Agents: Are they a good idea? *IBM Research Report,* IBM T. J. Watson Research Center.

HOHL, F. 1998. Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts, *Mobile Agent Security*, LNCS Vol. 1419, pp. 92-113, Springer-Verlag.

HOODA, A., KARMOUCH, A., AND ABU-HAKIMA, S. 1998. Nomadic Support Using Agent-level Communication, In *Proceedings of the 4th Symposium of Internetworking*, Canada.

HORVAT, H., CVETKOVI, D., MILUTNOVI, D., KOVI, P., AND KOVAEVI, V. 2000. Mobile Agents and Java Mobile Agent Toolkits, In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, IEEE Computing Society, Vol. 8, pp. 8029-8037.

IBM. 1999. Aglet Software Development Kit, Available at http://www.trl.ibm.co.jp/aglets/

JANSEN, W. & KARYGIANNIS, T. 2000. Mobile Agent Security, *NIST Special Publication 800-19.* Available at http://csrl.nist.gov/mobilesecurity / publications/sp 800-19.pdf [Accessed on 22/4/04].

JANSEN, W.A. 2000. Countermeasures for Mobile Agent Security, *Computer Communications*, Vol. 23, No. 17, pp. 1667-1676, Special issues on advance security techniques for network protection, Elsevier Science.

JANSEN, W.A. 1999. Mobile Agents and Security, *Web Report*. Available at: http://csrc.nist.gov/staff/Jensen/pp-agentsecurityfin.pdf. [Accessed on 12/3/04].

JOHANSEN D. 1999. Trend Wars: Mobile Agent Applications. *IEEE Concurrency*. Vol. 7, No. 3, pp 80-90.

JOHANSEN, D., LAUVSET, K. J., AND MARZULLO. K. 2002a. An Extensible Software Architecture for Mobile Components, In *Proceedings of the 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems*, pp. 231-237, Lund, Sweden.

JOHANSEN, D., LAUVSET, K. J., RENESSE, R. V., SCHNEIDER, F. B., SUDMANN, N. P., JACOBSEN, K. 2002b. A TACOMA retrospective, *Software-Practice and Experience*, Vol. 32, No. 6, pp. 605-619.

JOHANSEN, D., MARZULLO, K., SCHNEIDER, F., JACOBSEN, K. 1999. NAP: Practical Fault-tolerant for Itinerant Computations, In *Proceedings of the 19th International Conference on Distributed Computing Systems*, pp. 180-189.

JOHANSEN, D., VAN RENESSE, R., AND SCHNEIDER, F. B. 1995a. Operating System Support for Mobile agents, In *Proceedings of the 5th Workshop on Hot Topics in Operating Systems (HOTOS-V)*, pp. 42-45, IEEE press.

JOHANSEN, D., VAN RENESSE, R., SCHNEIDER, F. B. 1995b. An Introduction to the TACOMA Distributed System, *Computer Science Technical Report: 95-23*. Available at www.cs.uit.no/forskning/rapporter/Reports/9523.html

JUL, E., LEVY, H., HUTCHINSON, N., BLACK, A. 1988. Fine-grained mobility in the Emerald system, *ACM Transactions on Computer Systems*, Vol. 6, pp. 109-133.

KARJOTH, G., LANGE, D.B., AND OSHIMA M. 1997. The Aglet Security Model, *IEEE Internet Computing*. Vol. 1, No. 4, pp 68-77.

KARNIK, N., TRIPAHTI, A. 2001. Security in the Ajanta Mobile Agent System, *Software - Practice and Experience*, Vol. 31, No. 4, pp. 301-329.

KARNIK, N.M., AND TRIPATHI, A. 1998. Design Issues in Mobile-Agent Programming Systems, *IEEE Concurrency*. Vol. 6, No. 6, pp. 52-61.

KAUFMAN, C., PERLMAN, R., AND SPECINER, M. 1995. *Network Security: Private Communication in a Public World*, Englewood Cliffs, NJ: Prentice Hall, PTR.

KINIRY, J., AND ZIMMERMANN, D. 1997. A Hands-on Look at Java Mobile Agents, *IEEE Internet Computing*, Vol.1, No. 4, pp. 21-33.

KONE, M.T., SHIMAZU, A., NAKAJIMA, T. 2000. The state of the art in agent communication languages, *Knowledge and Information Systems*, Vol. 2, pp. 258-284.

KOTZANIKOLAOU, P., KATSIRELOS, G., AND CHRISSIKOPOULOS, V. 1999. Mobile Agents for Secure Electronic Transactions. *Recent Advances in Signal Processing and Communications*, pp. 363-368, World Scientific and Engineering Society Press.

LABROU, Y., FININ, T., AND PENG, Y. 1999. The current landscape of Agent Communication language, *IEEE Intelligent Systems*, Vol. 14, No. 2, pp. 45-52.

LANGE, D. B., OSHIMA, M. 1998. *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley, Reading, MA.

LAUVSET, K. J., JOHANSEN, D. AND MARZULLO, K. 2002. Factoring mobile agents. In *Proceedings of the 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems*, pp. 253-257, Alamitos, CA, USA.

LAUVSET, K. J., JOHANSEN, D., MARZULLO, K. 2001a. Separating Mobility from Mobile Agents, Available at http://www.cs.uit.no/forskning/rapporter/Reports /200139.ps. [Accessed on 2 September 2004].

LAUVSET, K. J., JOHANSEN, D., MARZULLO, K. 2001b. TOS: Kernel Support for Distributed Systems Management. In *Proceedings of the 2001 ACM Symposium on Applied Computing*, pp. 412-419. Las Vegas.

LEE, H., ALVES-FOSS, J., HARRISON, S. 2004. The Use of Encrypted Functions for Mobile Agent Security, *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS '04)*.

LEE, P., NECULA, G. 1997. Research on Proof-Carrying Code for Mobile-Code Security. Proceedings of the *Workshop on Foundations of Mobile Code Security*, Monterey. Available at www.cs.cmu.edu/~necula/fmcs 97.ps.gz. [Accessed on 14/2/2004].

LI, C., SONG, Q., AND ZHANG, C. 2004. MA-IDS Architecture for Distributed Intrusion Detection using Mobile Agents, In Proceedings of the *2nd International Conference on Information Technology for Application (ICITA 2004)* pp. 451-455. Available at http://attend.it.uts.edu.au/icita05/CDROM-ICITA04/papers/39-8.pdf [Accessed on 13/5/05].

LUC, M. 1999. Distributed directory service and message routing for mobile agents. *Technical Report ECSTR M99/3*, Department of Electronics and Computer Science, University of Southampton. Available at www.iam.ecs.solon.ac.uk/publications/ papers/paper3482.html [Accessed on 24/8/04].

MILOJICIC, D., BREUGST, M., BUSSE, I., CAMPBELL, J., COVACI, S., FRIEDMAN, B., KOSAKA, K., LANGE, D., OSHIMA, M., THAM, C., VIRDHAGRISWARAN, S., WHITE, J. 1998. MASIF: the OMG mobile gent system interoperability facility, In *Proceedings of the Second International Workshop on Mobile Agents,* Lecture Notes in Computer Science, Vol. 1477, pp. 50-67, Springer-Verlag, Berlin.

MONTANARI, R., STEFANELLI, C., AND NARANKER, D. 2001. Flexible security policies for mobile agent systems. *Microprocessors and Microsystems*, Vol. 25, pp. 93-99.

NECULA, G., AND LEE, P. 1996. Safe Kernel Extensions without Run-Time Checking", In *Proceedings of 2nd Symposium on Operating System Design and Implementation (OSDI '96)*, pp. 229-243, Seattle, Washington.

OMICINI, A., AND DENTI, E. 2001. From tuple space to tuple centers, *Science of Computer Programming*, Vol. 41, pp 277-294, Elsevier.

ORGANIZATION FOR ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS). 2003. Assertion and Protocol for the OASIS Security Assertion Markup Language 3 (SAML) V1.14, Available at http://www.oasis-open.org/ committees/download.pdf/1894/sstc-saml-core-1.1-draft-10.pdf. [Accessed on 27/3/04].

Orso, A., Vigna, G., Harrold, M. 2001. MASSA: Mobile Agents Security through Static/Dynamic Analysis, In Proceedings of the ICSE Workshop on Software Engineering and Mobility, Ontario Canada. Available at http://www.cc.gatech.edu/~orso/papers/orso.vigna.harrold.IWSEM01.pdf [Accessed on 2/5/05].

PAGE, A., ZASLAVSKY, A., INDRAWAN, M. 2003. Evaluating Security in Software Agent Systems using a Secure Analysis Tool, In *Proceedings of the 1st Australian Information Security Management Conference*,

PAGNIA, H., VOGT, H., GÄRTNER, F., WILHELM, U. 2000. Solving Fair Exchange with Mobile Agents In: Kotz, D.; Mattern, F. (Eds.): Agent Systems, Mobile Agents, and Applications. *Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents, ASA/MA 2000*, LNCS 1882, pp. 57-72. Springer-Verlag.

PAPAIOANNOU T. 1999. Mobile Agents: Are They Useful for Establishing a Virtual Presence in Space? In *Agents with Adjustable Autonomy Symposium*, part of the AAAI 1999 Spring Symposium Series. Available at http://www.luckyspin.org/Docs/ [Accessed on 22/7/04].

PEINE, H. 1997. An Introduction to Mobile Agent Programming And The Ara System, *Technical report ZRI-Report 1/97*, Department of Computer Science, University of Kaiserslautern, Germany. Available at http://wwwagss.informatik. uni-kl.de/ projekte/Ara/Doc/intro-prog.ps.gz. [Accessed on 2/6/04].

PEINE, H., AND STOLPMANN, T. 1997. The Architecture of the Ara Platform for Mobile Agents, Mobile Agents, In K. Rothermel and R. Popescu-Zeletin edition, *Mobile Agents: First International workshop MA'97*, Vol. 1219, pp. 50-61, Springer. Berlin.

PEINE, H. 2002. Application and Programming Experience with the Ara Mobile Agent System, *Software- Practice and Experience*. Vol. 32, No. 6, pp. 515-541.

PEINE, H. 1997. Ara - Agents for Remote Acton, In *Itinerant Agents: Explanations and Examples with CD-ROM*, W. Cockayne and M. Zyda (ed), Manning/Prentice Hall.

PEINE, H. 1998. Security concepts and Implementations on the Ara Mobile Agent System, In *Proceedings of the seventh IEEE Workshop on Enabling Technologies: Infrastructure for the Collaborative Enterprises*, WETICE'98. California, USA. Available at http://wwwagss.informatik.uni-kl.de/ projekte/Ara/Doc/ara-security.ps.gz

PHAM V. A., KARMOUCH A. 1998. "Mobile Software Agents: An Overview", *IEEE Communications Magazine*. Vol. 36, No. 7, pp. 26-37.

PICCO, G P., 2001. Mobile agents: an introduction, *Microprocessor and Microsystems*, Vol. 2, No. 2, pp. 65-74.

PINSDORF, U., AND ROTH, V. 2002. Mobile Agent Interoperability Patterns and Practices, In *Proceedings of Ninth IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, pp. 238-244, Sweden.

PITOURA, E. AND FUDOS, I., 2001. Distributed Location Databases for Tracking Highly Mobile Objects, *The Computer Journal*, Vol. 44, No. 2 pp. 75-91.

PLEISCH, S., AND SCHIPER, A. 2003. Fault-tolerant mobile agent execution, *IEEE Transactions on Computers*, Vol. 52, No. 2, pp 209-222.

RAO, J., MIHHAIL S. X. 2003. Implementation Explanation Ontology for Agent Systems, In *Proceedings of the IEEE/WIC International Conference on Web Intelligence*, Canada.

ROTHERMEL, K., AND SCHWEHM, M. 1998. Mobile Agents, *Encyclopaedia for Computer Science and Technology*, New York: M. Dekker Inc.

ROVATSOS, M. AND NICKLES, M. AND WEIß, G. 2004. An empirical model of communication in multiagent systems. *Lecture Notes in Computer Science*, 2922.

SAMEH A., AND FAKHRY, D. 2002. Security in Mobile Agent Systems, *Proceedings of the 2002 symposium on Applications and the Internet (SAINI '02)* Available at http://csdl.computer.org/comp/proceedings/saint/2002/1447/00/14470004.pdf [Accessed on 28/1/2005].

SANDER, T. AND TSCHUDIN, C.F. 1998. Protecting Mobile agents against Malicious Hosts, *Mobile agent security*, LNCS Vol.1419, pp. 44-60, Springer-Verlag.

SATOH I. 2001. Adaptive Protocols for Agent Migration, *Proceeding of the 21st International Conference on Distributes Systems (ICDCS-21)*, pp. 711-714. Available at http://research.nii.ac.jp/~ichiro/papers/satoh-icdcs2001.pdf [Accessed on 28/1/2005]

SCHELDERUP, K., AND OLNES, J. 1999. Mobile Agents Security- Issues and Directions, *Lecture Notes in Computer Science*, Vol. 1579, pp. 155-167.

SCHOEMAN, M., CLOETE, E. 2003. Architectural Components for the Efficient Design of Mobile Agent Systems, In *Proceedings of SAICSIT*, pp. 48-58.

SIERRA, WOODRIDGE, ADEH. 2000. Agent Research and Development in Europe. *IEEE Internet Computing*, pp.81-83.

STALLING, W. 1995. *Network and Internetwork security: Principle and Practice*, Prentice-Hall, Englewood Cliffs, NJ.

STRAβER, M., BAUMANN, J., HOHL, F. 1996. Mole-A Java Based Mobile Agent Systems, *Special Issue Object-Oriented Programming: Workshop Reader of the 10th European Conference Object-Oriented Programming ECOOP'96*, pp. 327-334.

STRAβER, M., ROTHERMEL, K. 1998. A Fault- tolerance protocol for providing the exactly-once property of mobile agents, In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems (SRDS'98)*, pp. 100-108, Los Alamitos, CA, USA.

SUDMANN, N. P. AND JOHANSEN, D. 2000. Adding Mobility to Non-mobile Web Robots. In *Proceedings of the workshop on Knowledge Discovery and Data Mining in the World-Wide Web at the 20th IEEE International Conference on Distributed Computing Systems*, Taiwan. Available at http://www.cs.uit.no/forskning/rapporter/Reports/200036.ps. [Accessed on 2/9/04].

SUDMANN, N. P., JOHANSEN D. 2001. Building agent applications using wrappers. Available at *http://www.cs.uit.no/forskning/rapporter /Reports/200138.ps*. [Accessed on 2/9/04].

SUTHERLAND, D. 1997. RMI and Java Distributed Computing. Available at *http://java.sun.com/features/1997/nov/rmi.html*. [Accessed on 15/4/2005].

SYCARA, K., GIAMPAPA, J.A., LANGLEY, B. K., AND PAOLUCCI, M. 2003. "The RETSINA MAS, a Case Study," *Software Engineering for Large-Scale Multi-Agent Systems: Research Issues and Practical Applications*, LNCS 2603, pp. 232-250, Springer-Verlag, Berlin.

TAHA, K., AND PILIOURA, T., Agent Naming and Locating: Impact on Agent Design. Available at http://cui.unigr.ch/OSG/publications/OO-articles/TechnicalReports/99/AgentNamingLocatin.pdf [Accessed on 22/6/04].

TAI, H., KOSAKA, K. 1999. The Aglet Project, *Communications of the ACM*, Vol. 42, No. 3, pp. 100-101.

TAN, H. K., MOREAU, L., 2002. Certificates for Mobile Code Security, *ACM Symposium on Applied Computing*, pp. 76-81.

TAUBER, J. 1999. "XML After 1.0: You Ain't Seen Nothing Yet," *IEEE Internet Computing*, Vol. 3, No. 3, pp. 100-102.

THORN, T. 1997. Programming language for mobile code, *ACM Computing Surveys*, Vol. 29, No. 3, pp. 213-239.

TRIPATHI, A.R., AHMED, T. & KARNIK N. M. 2001. Experiences and future challenges in mobile agent programming. *Microprocessor and Microsystems*, Vol. 25, pp.121-129.

TRIPATHI, A.R., KARNIK, N.M., AHMED, T., SINGH, R.D., PRAKASH, A., KAKANI, V., VORA, M.K., PATHAK, M. 2002. Design of the Ajanta System for Mobile Agent Programming, *The Journal of System and Software*, Vol. 62, pp.123-140.

UHRMACHER, A. M., KULLICK, G. B. 2000. Plug and Test - Software Agents in Virtual Environment, *Proceedings of the 2000 Winter simulation Conference*, pp 1722-1729.

VIGNA, G. 1998. Cryptographic traces for Mobile agents, G. Vigna edition, *Mobile Agents and Security*, LNCS 1419, Springer-Verlag, Berlin.

VITEK J., TSCHUDIN C. 1997. Security and communication in Mobile Objects Systems. J. Vitek and C Tschudin, Eds., *Mobile object Systems: Towards the programmable Internet*, LNCS, Vol. 1222, Springer.

WAGNER, D. AND SCHNEIER, B. 1996 Analysis of the SSL 3.0 protocol, *In Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, Oakland, USA, November, pp. 29-40. Available at: http://www.usenix.org/publications /library/proceedings/ec96/wagner.html [Accesses on 10/5/04].

WALLIN, A. 2004. Secure Auction for mobile agents, ESPOO Available at http://www.vtt.fi/inf/pdf/publications/2004/P538.pdf [accesses on 12/5/05] .

WAYNER, P. 1995. Free Agents. *Byte (March Edition)*.

WHITE J. E. 1996. Telescript Technology: Mobile Agents, J.Bradshaw (ed.) *Software Agents*, AAAI Press/MIT Press.

WIES, R. 1995. Using a Classification of Management Policies for Policy Specification and Policy transformation, In *Proceedings of ISINM'95*, pp. 44-56, Chapman & Hall.

WILHELM, U., STAAMANN, S., AND BUTTYAN, L. 1998. On the problem of trust in mobile agent systems, *Network and Distributed System Security Symposium*, San Diego, CA. Available at http://citeseer.ist.psu.edu/ wilhelm98problem.html [Accessed on 5/5/04].

WITTNER, O., 1999. Evaluation of Mobile agent systems with respect to failure semantics, Available at http://www.item.ntun.no/~witner/papers/failsemMASreport.pdf [Accessed on 18/1/2005]

WOLFGANG, E. 2000. *Engineering Distributed Objects.* John Wiley & Sons.

WONG D., PACIONREK N., WALSH T., DICELIA J., YOUNG M., PEET B. 1997. Concordia: An Infrastructure for Collaborating Mobile Agents, In *Proceedings of the first International Workshop Mobile Agents, LNCS 1219*, pp. 86-97, Springer-Verlag.

WOOLDRIDGE, M. J., AND JENNINGS N R. 1999. Software Engineering with Agents: Pitfalls and Pratfalls, *IEEE Internet Computing*, Vol. 3, pp. 20-27.

WOOLDRIDGE, M. J. 2000. Semantic issues in the verification of agent communication languages, *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 3, No. 1, pp 9-31.

XUDONG G., YILING Y., JINYUAN Y. 2000. POM - A Mobile Agent Security Model against Malicious Hosts**,** *The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region*, Vol. 2**,** No. 5 pp 14 – 05.

XUHUI, L., JIANNONG, C., YANXIANG, H. 2004. A direct Execution approach to simulating Mobile agent Algorithms, *The Journal of Supercomputing*, Vol. 29, No. 2, pp. 171-184.

YAP, M. T., HENG, C. K., WONG, K. K., LEONG, P. C. 2004. Subject Bidding Through Mobile Agents, *Proceedings of the 2nd International Conference on Information Technology for Application (ICITA 2004)*, pp56-60.

YEN, J., FAN, X., VOLZ, R. 2004. Proactive Communications on Agent Teamwork, *Advances in Agent Communication: International workshop on Agent Communication languages, ACL* , LNAI, Vol. 2922, pp 271-290.

YI, X., WANG, X. F., AND LAM K. Y. 1998. A secure Intelligent Trade Agent System. In *Proceedings of the International IFIP/GI Working Conference, TREC'98*, LNCS Vol. 1402, pp. 218-228, Springer-Verlag.

ZASLAVSKY, A. 2004. Mobile Agents: Can They Assist with Context Awareness? *Proceedings of the IEEE International Conference on Mobile Data Management (MDM'04)*, Available at http://csdl.computer.org/comp/proceedings/mdm/2004/2070/00/20700304.pdf [Accesses on 27/1/2005]

Table of Content

6 Evaluations Of The Selected Mobile Agent Systems 68

Error! No table of figures entries found.