Issues on Mobile Agent Technology Adoption

E.P. Ebietomere and G.O. Ekuobase

1Department of Computer Science
University of Benin
Benin City, Edo State, Nigeria
princewill.ebietomere@uniben.edu; godspower.ekuobase@uniben.edu

1Corresponding Author

ABSTRACT
The design, development, deployment and use of mobile agent in computing is a relatively new paradigm in distributed systems environment. The deployment of mobile agent in creating distributed systems has been shown to help reduce network load, overcome network latency and encapsulate protocols as well as help the dynamic adaptation, and asynchronous and autonomous execution of distributed applications. Despite these benefits, mobile agent technology adoption is still very low even in the face of high research activities in the area and this is a serious hindrance to tapping the computational benefit of mobile agent technology. This paper explicates the nitty-gritty behind mobile agent technology and exposed the factors militating against its adoption. This will help expose critical areas one need to focus on in mobile agent research to increase the adoption of mobile agent technology.

Keywords: Software agent, Mobile agent, Technology adoption and Distributed application

African Journal of Computing & ICT Reference Format:
E.P. Ebietomere and G.O. Ekuobase (2014). Issues on Mobile Agent Technology Adoption

1. INTRODUCTION

A mobile agent is a software agent, enabled with the capability of hopping or migrating from one host to another via a network, and can suspend execution in one location and resume at another location in carrying out task(s) on behalf of its owner [2, 3, 8, 15]. Mobile agent technology involves the design, development, deployment and use of mobile agents in computing. Mobile agent technology gained popularity in the nineties when research in the area was very active due to paradigm shift from standalone applications to distributed applications as a result of the advent of the internet.

The deployment of mobile agent in creating distributed systems has been shown to help reduce network load, overcome network latency and encapsulate protocols as well as the dynamic adaptation, and asynchronous and autonomous execution of distributed applications [1, 4, 10, 17, 21]. Other benefits include: support for heterogeneity, robustness and fault-tolerance. Applications such as e-commerce, personal assistance, secure brokering, distributed information retrieval; telecommunication network services, workflow applications and groupware, monitoring and notifications, information dissemination, and parallel processing are already enjoying the benefits of mobile agent technology [1, 10, 21]. Despite the potentials of mobile agent technology, its use is dwindling so much so that some of the use of some systems that were built on mobile agent technology has been discontinued over the years [15, 16, 21]. What is more worrisome is that the low adoption of mobile agent persists even when the research domain is increasingly active [21].

What could be responsible for the low adoption of mobile agent technology considering its immense potentials and high research activity? This question is particularly important considering the fact that the research effort into mobile agent technology may be futile if its adoption problems are not addressed; for a technology that cannot be used no matter how beautiful it may appear is as worthless as it not being in existence. This is the essence of this paper, to account for the low adoption of mobile agent technology.

2. BACKGROUND INFORMATION

Mobile agent technology is a relatively new paradigm in distributed systems environment. An approach believed, will replace the normal client-server model and its architecture [1]. Several definitions have been ascribed to mobile agents; some of them are presented next. A mobile agent can be seen as a program that is goal-directed and capable of suspending execution on one platform where it can resume execution. That is to say, a mobile agent exercises its owner’s authority, work autonomously towards a goal and can also meet and interact with other agents [8]. A mobile agent is also seen as a particular class of software agent that can migrate during execution from one host to another where it can resume execution. This is to say, a mobile agent exercises its owner’s authority, work autonomously towards a goal and can also meet and interact with other agents [8]. A mobile agent is also seen as a particular class of software agent that can migrate during execution from one host to another where it can resume execution [21]. Mobile agent can also be seen as an autonomous software entity with the capability of roaming among nodes in a network-aware fashion. It can move from host to host to find the needed resources [4].
[21] described mobile agent as a process that can move its state from one environment to the other holding its data intact as they migrate and can perform appropriately in the new environment.

2.1 Characteristics of Mobile Agents
A mobile agent is a software agent. A software agent differs from other programs because of the following properties [7, 17]:

- **Intelligence**: a software agent employs a technique from the field of artificial intelligence which makes it possible for them to act and perform task(s) on behalf of their owners in such a way that one would think that the tasks were carried out by human. That is to say, they can actually act as if they have intelligence.

- **Autonomy**: this is the ability of the agent to exercise control over its own actions. It can decide the sequence of action to perform in order to achieve the user’s task once specifications are given to the agent.

- **Responsiveness**: this is the ability of an agent to perceive its environment and respond in real-time to changes in the environment. Apart from being reactive to their environment they are also goal-oriented (proactive and purposeful), taking advantage of opportunities when necessary.

- **Communicative ability**: an agent does not work in isolation. They are sociable entities and as such interact and collaborate with other software agents in carryout their tasks. For this reasons, agents should provide a friendly interface to allow for easy interactivity with other agents.

- **Adaptability**: agents learn about user’s behaviour and adjust to changes based on previous experience.

- **Ragged**: mobile agents usually have the ability to recover from errors whenever they occur.

From the definitions of mobile agents, one can see it combines the properties of a software agent and “mobility” as a property; i.e. MAp = SAP + Mobility. Where MAp is the properties of a Mobile Agent and SAP is the properties of a Software Agent.

The mobility exhibited by a mobile agent could either be weak or strong [20]. We say it is weak if the mobile agent transports or migrates itself (code) with its data and if it transports or migrates itself with its data and state, we say it is strong. Thus we write:

- Weak mobility = mobile agent code + data
- Strong mobility = Weak mobility + state

A mobile agent code is the set of instructions describing the execution of the agent while data is an information storage area used by the agent. Results of executions are usually stored in this area. This part of the agent is reserved for storing information regarding the state of the agent.

2.2 Mobile Agent Life Cycle
Mobile agents go through some processes to get job done. The entire process is termed Mobile Agent Life Cycle; it is depicted in figure 2.1 and highlighted as follows [17, 21]:

- **Creation**: this is the first phase of a mobile agent life cycle. Once a request is made to a mobile agent, an instance of the mobile agent is created and its state is initialized.

- **Dispatch**: this involves the movement of the mobile agent from one node to the other and can be achieved by specifying the address of the destination.

- **Cloning**: this refers to creating a copy of the original mobile agent object. That is to say, a twin agent is born and the current state of the original is duplicated in the clone.

- **Deactivation**: a mobile agent is put to sleep and its state is stored on a disk of the host.

- **Activation**: a deactivated mobile agent is brought back to life and its state is restored from disk.

- **Retraction**: an agent is brought back from a remote host along with its state to the home machine after the completion of its job.

- **Disposal**: this is done at the end of the mobile agent life cycle. The agent is terminated and its state is lost forever.
2.3 Mobile Agent Model
A mobile agent must contain all of the following models: an agent model, a life cycle model, a computational model, security model, a communication model, and a navigation model [19].

2.3.1 Agent Model
This model defines the internal structure of the intelligent part of the agent [13]. The structure of this model actually defines the autonomy, learning and cooperative characteristics of an agent. Besides, it specifies the reactive and proactive nature of the agent.

2.3.2 Life-cycle Model
This model defines different execution states of a mobile agent and the events that caused the movement from one state to another. Prominent amongst life cycle models are the persistence process model adopted by Telescript and agent TCL and the task based model adopted by Aglets [5, 25]. When a mobile agent is transported from one node to the other, the process in the running state is check-pointed and the agent enters what is termed “frozen” state. Next, the context is delivered to the destination mode where the process is resumed and re-enters the “running” state at the point it left off.

2.3.3 Computational Model
The computational model refers to the computational capabilities of an agent, which include data manipulation and thread control primitives. That is to say, computational model defines how a mobile agent executes when it is in a “running” state. This model takes place in an environment and it is facilitated by some form of processor. A processor could be the CPU of the computer or more abstract processor as can be found in Java virtual machine. Besides, implementers of mobile agents gain access to other models via the computational model; hence its structure affects all other models.

2.3.4 Security Model
The security model describes the ways in which agents can access network resources, as well as the ways of accessing the internals of the agents from the network. This model is very important because a mobile agent system is an open system and just like every other open systems, it is subject to different attacks. There are three fundamental security issues specific to mobile agent systems [10]. These are:

- Protecting the host (platform) from the mobile agent,
- Protecting the mobile agent from other mobile agents, and
- Protecting the mobile agent from the host.

2.3.5 Communication Model
The importance of this model cannot be over-emphasized. Without this model, interaction with other entities in a computing environment will not be possible. Communication can be synchronous, asynchronous or deferred. Communication is synchronous when the agents need to interact with one another in real time with the communicating agents exclusively bound to one another and when they are not
exclusively bound in real time during communication we say communication is asynchronous. Deferred communication [12, 23] is a hybrid model, in which the sending agent continues with other activity until the desired results are available. Communication is used when there is a need to access services outside of the mobile agent for the sake of cooperation and coordination.

There are quite a number of languages for communication and coordination of agents referred to as Agent Communication Languages (ACLs). Table 2.1 enumerates and describes the most prominent examples of ACLs as in [24].

Table 2.1: Most prominent Agent Communication Languages [24].

<table>
<thead>
<tr>
<th>Agent Communication Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Knowledge Query and Manipulation Language (KQML)</td>
<td>It is perhaps the most widely used agent communication language (Weij, 2002).</td>
</tr>
<tr>
<td>2. ARTIMIS Communication Language (ARCOL)</td>
<td>It is the communication language used in the ARTIMIS system. ARCOL has smaller set of communication primitives than KQML, but these can be composed.</td>
</tr>
<tr>
<td>3. FIPA Agent Communication Language (FIPA-ACL)</td>
<td>It is an agent communication language largely influenced by ARCOL. Together FIPA-ACL, ARCOL and KQML establish a quasi-standard for agent communication languages.</td>
</tr>
<tr>
<td>4. Knowledge Interchange Format (KIF)</td>
<td>It is a logic based language that has been designed to express any kind of knowledge and non- knowledge. KIF is a language for content communication whereas languages like KQML, ARCOL and FIPF-ACL are for intention communication.</td>
</tr>
<tr>
<td>5. Domain independent COOrdination Language (COOL)</td>
<td>It aims at explicitly representing and applying coordination knowledge for multi-agent systems and focuses on rule-based conversation management. Language like COOL can be thought of as supporting a coordination/communication (or protocol-sensitive) layer above intention communication.</td>
</tr>
</tbody>
</table>

2.3.6 Navigation Model
This model concerns itself with all aspects of agent mobility from the discovery and resolution of destination hosts to the manner in which a mobile agent is transported. This concept may not be evident in agents that execute remotely without the need to meet other agents at a place, but for agents that execute locally where meeting at a place which could either be in a shop or home is of importance, the navigation model is paramount.

2.4 Implementation of Mobile Agents
There are different languages for implementing mobile agents. Some languages such as obliq and telescript have been designed specifically for writing mobile agents. There are also many mobile agent technology and platforms (see Table 2.2) implemented in general purpose languages with an extended special library.

Table 2.2: Mobile Agent Systems [18]

<table>
<thead>
<tr>
<th>Mobile Agent</th>
<th>Language</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ajanta</td>
<td>Java</td>
<td>University of Minnesota, U.S.A</td>
</tr>
<tr>
<td>ARA</td>
<td>Tcl, C, Java</td>
<td>University of Kaiserslautern, Germany</td>
</tr>
<tr>
<td>Agent Tcl</td>
<td>Tcl</td>
<td>Dartmouth College, U.S.A</td>
</tr>
<tr>
<td>Aglets</td>
<td>Java</td>
<td>IBM, Japan</td>
</tr>
<tr>
<td>Concordia</td>
<td>Java</td>
<td>Mitsubishi, U.S.A</td>
</tr>
<tr>
<td>CyberAgents</td>
<td>Java</td>
<td>FTP Software Inc., U.S.A</td>
</tr>
<tr>
<td>fMAIN</td>
<td>Tcl, Perl, Java</td>
<td>University of Frankfurt, Germany</td>
</tr>
<tr>
<td>FIPA-OS</td>
<td>Java</td>
<td>Emorphia, UK</td>
</tr>
<tr>
<td>Grasshopper</td>
<td>Java</td>
<td>IKV, U.S.A</td>
</tr>
<tr>
<td>Java-to-go</td>
<td>Java</td>
<td>University of California Berkeley, U.S.A</td>
</tr>
<tr>
<td>Kafka</td>
<td>Java</td>
<td>Fujitsu, Japan</td>
</tr>
<tr>
<td>Messengers</td>
<td>M0</td>
<td>University of Zurich, Switzerland</td>
</tr>
<tr>
<td>MOA</td>
<td>Java</td>
<td>The Open Group, U.S.A</td>
</tr>
<tr>
<td>Mole</td>
<td>Java</td>
<td>University of Stuttgart, Germany</td>
</tr>
<tr>
<td>Monja</td>
<td>Java</td>
<td>Mitsubishi, Japan</td>
</tr>
<tr>
<td>Odyssey</td>
<td>Java</td>
<td>General Magic, U.S.A</td>
</tr>
<tr>
<td>SOMA</td>
<td>Java</td>
<td>University of Bologna, Italy</td>
</tr>
<tr>
<td>TACOMA</td>
<td>Tcl, C, Python</td>
<td>Cornell, U.S.A i Tromso, Norway</td>
</tr>
<tr>
<td>Telescript</td>
<td>Telescript</td>
<td>General Magic, U.S.A</td>
</tr>
<tr>
<td>Voyager</td>
<td>Java</td>
<td>Object Space Inc., U.S.A</td>
</tr>
</tbody>
</table>
Some of these languages are described as follows:

2.4.1 Obliq
This is a lexically scoped, un-typed interpreted language that supports distributed object computation. The language supports objects and not classes. It uses the prototype based model of object oriented programming. Obliq uses runtime type checking. It has built-in procedures for importing and exporting procedures and objects between machines.

2.4.2 Java
The Java environment is an approach to distributed computing. It is a general purpose language. It is an object oriented language. Java programs are compiled to Java byte codes, binary instructions for the Java Virtual Machine. More so, Java programs can run on any platform with JVM interpreter which makes its programs highly portable. Though the language was not designed for writing mobile agents, it has the necessary capabilities for mobile agent programming which has made it a basis for many implementations of mobile agent systems. Most of the systems make use of Java RMI - Remote Method Invocation. For example Aglet- uses an event driven approach to mobile agents that is analogous to the Java library Applet class [22].

2.4.3 Perl 5
Penguin is a Perl 5 module with functions enabling the sending of Perl scripts to a remote machine for execution and for receiving perl scripts from remote machines for execution. The scripts are digitally signed to allow authentication and are executed in a secure environment. Mobile agents written in Perl are restricted in that they must always restart execution at the same point. There is also no support for agents saving their state on migration. A new Agent Module v3.0 is being created to give Perl 5 more sophisticated mobile agent capabilities. The additional features include giving agents the ability to save their state on migration.

2.4.4 Python
Python is an object-oriented scripting language. The Corporation for National Research Institution, USA uses Python as a language for implementing Knowbot programs. It is also one of the languages used in developing Tacoma.

2.4.5 Telescript
It is a Remote Object Oriented mobile agent language. It claims to be “a platform that enables the creation of active, distributed network applications”. Its main achievement is in its use in the Magic Link Project, Magic cap product and electronic market places. Telescript has three main concepts; agents, places and the “go” instruction. Agents travel from places to places using the “go” instruction. They can interact with each other and any services located at the places visited through the use of “meet” instruction.

Telescript is a whole new platform independent system consisting of a language and an interpreter called the Telescript Engine. The computational model is the Telescript language itself and the agent model is simply a Telescript object. Special instructions implement part of the other models. The navigational model is implemented by the concept of places and the “go” instructions. The “meet” and “connect” instructions are the basis of communication model. Finally, the security model is implemented with the authority, region and identity concepts. In general, any language used for writing a mobile agent must support some of the following [22]: agent migration, communication between agents, access to server resources, security mechanisms, appropriate efficiency, the ability to run on multiple platforms, and ease of writing mobile agent application. Besides, the degree to which a language can support the above listed characteristics determines its level of suitability (usefulness) for writing mobile agent applications.

2.5 Agent Systems
Agent systems are implemented through agent platforms, the underlying software for the system. Platform defines a standard around which a system can be developed. Platform may provide support for migration, naming, location and communication services. Examples of agent platforms are as shown in table 2.2. Furthermore, an agent system consists of one or more execution environments or agencies. How an agency is constituted or built is dependent on the type of agent system one is building, but a generic architecture is as shown in figure 2.2.
2.6 Benefits of Adopting Mobile Agent Technology

There are at least seven good reasons why mobile agents should be used [10]. They are highlighted as follows:

i. They reduce network load - distributed systems often rely on communication protocols involving multiple interactions to accomplish a given task. This usually results in a lot of network traffic. Mobile agents allow users to package conversations and dispatch it to a destination host where interactions take place locally. Also, load in a network may be high as a result of movement of large data via the network as could be seen in a normal client server approach. The use of mobile agent can help ameliorate this because computation is moved to data rather than data been moved to computation.

ii. They overcome network latency – in critical environment such as real time where time of response to change is a crux, controlling such systems through a potentially large network involves significant latencies which for critical real-time systems are not tolerable. Mobile agents offer a solution because they can be dispatched from a central controller to act locally and execute the controller’s direction directly.

iii. They encapsulate protocols – when data is exchanged in a distributed system, each host owns the code that implements the protocols needed to properly code outgoing data and interpret incoming data. However, as protocols evolve to accommodate new requirements for efficiency or security, it is cumbersome if not impossible to upgrade protocol code properly. As a result, protocols often become a legacy problem. Mobile agents on the other hand can move to remote hosts to establish channels based on proprietary protocols.

iv. They execute asynchronously and autonomously – tasks requiring continuously open connection between a mobile device and a fixed network are probably not economically or technically feasible. To solve this problem, tasks can be embedded into mobile agents, which can then be dispatched into the network. After being dispatched, the agents become independent of the process that created them and can operate asynchronously and autonomously.

v. They adapt dynamically – mobile agents can sense their execution environment and react autonomously to changes. Multiple mobile agents have the unique ability of distributing themselves among the hosts in the network to maintain optimal configuration for solving particular problem.

vi. They are naturally heterogeneous – network computing is basically heterogeneous from both hardware and software perspective. Mobile agents provide optimal conditions for seamless system integration because they depend only on their execution environments and independent of hardware and transport layer.
vii. They are robust and fault tolerant – mobile agents’ ability to react dynamically to unfavourable situations and events makes it easier to build robust and fault tolerant distributed systems. If a host is being shut down, all agents executing on that machine are warned and given time to dispatch and continue their operation on another host in the network.

2.7 Trends in Mobile Agent Technology

The emergence of the internet brought about a shift in paradigm from building standalone applications (i.e. desktop application) to building distributed applications. This shift in paradigm was accompanied by problem of network load and latency, heterogeneity and network failure problems and in solving these problems; a new technology in the name mobile agent was born.

Mobile agent technology did not emerge from the oblivion rather it came to being through evolution. In the first step of the evolution, we find the mobility of files, for example with the FTP protocol. After this, there was the Remote Procedure Call (RPC) in which case data is moved between a client and a server; as depicted in figure 2.3.

Next was Remote Evaluation (REV). This is in all ways similar to RPC, the only difference being that code is the parameter transported instead of data. After REV came Code on Demand (COD). In the code on demand (COD) paradigm, the computational component, client has local access to the resources, but does not know how to execute the task. Thus, it contacts a computational component, server, on a different site, which provides the know-how. The client loads the know-how from the server and executes the task locally, as shown in figure 2.4.

Finally, active entities were able to change the environment where they are executing. Thus, the natural evolution of mobility resulted in code mobility. New technologies usually go through stages termed “life-cycle”. A model of life-cycle of a technology could be seen as pictorially described by [14] in figure 2.5. Mobile agent technology holds no exception to this. It is important to note that mobile agent technology is still at the irruption stage [11] due to its poor adoption [21] though one probably would have thought that by now the adoption of the technology should have grown past this stage of irruption.
Attempt would be made in the next session to explicate factors that has kept mobile agent at the irruption stage. This will help expose how we may improve the level of adoption of mobile agent technology.

3. FACTORS MILITATING AGAINST THE ADOPTION OF MOBILE AGENT TECHNOLOGY

Various factors have been identified to be responsible for the low adoption of mobile agent technology. These factors include language of implementation of the agent, coordination model, security, efficiency and standardization [1, 4, 9, 8, 16, 18, 21].

3.1 Language of Implementation

There are different languages for implementing mobile agents. Languages such as Telescript and Obliq were built specifically for implementing mobile agents while languages like Java, Python, Tcl, and Perl 5 amongst others were not but have capabilities which make them applicable in the domain. The challenge here is the need for compatibility of the agents with their agencies. Agency is the environment in which an agent carries out its activities.

Every language for implementing mobile agents imposes a constraint which in the long run affects the way the agents behave in its running environment. It is therefore important for builders of agent systems to take into consideration the platform and environment in which the agent is to carry out its activities and put in place machinery that will allow for compatibility. However, Java is gradually becoming a de facto language despite its support for weak mobility.

This is as a result of its support for platform independence. This blurs the need of having to know about the agency on which these agents will run. It is therefore pertinent to painstakingly weigh all pros and cons of our choice of language of implementation of the mobile agents to be sure that the desired and the most appropriate in a given scenario is used. This undoubtedly will help guarantee the sustainability of the system.

3.2 Coordination

Coordination is another factor seriously affecting the adoption of mobile agent technology. Agents are usually not built to operate in isolation but to interact with other agents in a distributed environment. If agents must interact with one another, then a mechanism must be in place to ensure proper coherence of actions amongst them. This mechanism that ensures effective interaction amongst agents defines coordination. The challenge really is in identifying these mechanisms that allow agents to coordinate their activities or actions automatically without the need for human supervision [4]. When coordination is carried out amongst agents with a common goal in mind, it is referred to as cooperation. If mobile agents must achieve their purpose in a distributed environment, then the issue of coordination of these agents must be given a proper attention otherwise, its adoption rate will continue to plummet.
3.3 Standardization
Standardization is one very important issue that cannot be
thrown out in the field of mobile agent technology. Agents are
built by different persons, laboratories and companies with
different philosophies. Agents’ environment by nature is
heterogeneous and for entities in this domain to interact
appropriately, there is a need for standardization at least in the
area of communication, coordination and transportation [1].
That is to say, how agents communicate with one another and
other entities, how they are coordinated to achieve their goals
and how they navigate. There are currently two standards for
mobile agent technology.

The Object Management Group’s Mobile Agent System
Interoperability Facility (MASIF) - managed by the Object
Management Group with CORBA being its most famous
standard and the other is the specification managed by
Foundation for Intelligent and Physical Agent (FIPA). OMG
enforces such standards as making sure messages are
schedulable as well as event driven, support of transportation
mechanism for unique addressing as well as role-based
addresses and support of transportation mechanism for unicast
and broadcast modes of mobile agents.

MASIF basically provides interface between agent and agent
systems and it is limited to the interworking of agent systems
using same language – a serious barrier to interoperability of
mobile agent in a heterogenous platform while the strength of
FIPA is on communication among agents and not in mobility
i.e. FIPA is weak transportation standards which is however
necessary for the mobility of agents [18]. Obviously,
therefore, we need uniform standard that will guarantee the
standardization requirements of mobile agents –
communication, coordination and transportation; as exposed
by [1]. Until this is done and enforced, we may continue to
experience low adoption of mobile agent technology.

3.4 Security
This is one aspect of mobile agent technology whose
importance cannot be over-emphasized. Security is the key
reason why mobile agent adoption is low [17]. One may of
course argue that despite the security challenges in some
technological domains, the adoption of those technologies is
still high and as such, mobile agent technology should not be
an exception. Unfortunately, security as it affects mobile
agents, stems from two directions which complicate the
security problem of mobile agent. The first is as a result of the
properties exhibited by mobile agents and the second stems
from the fact that mobile agents dwells in an open system. The
higher the degree of mobility, autonomy and execution of an
agent, the more vulnerable a mobile agent is to security threat
[15]. As earlier stated, [10] exposed three fundamental issues
specific to mobile agent systems as an open system. They are:
protecting the host from the agent, protecting agent from
another agent and protecting the agent from a host; as
discussed in the following subsections.

3.4.1 Protecting the Host from the Mobile Agent
This could be in the form of masquerading in which case an
unauthorized agent claims the identity of another agent. The
masquerading agent may pose as an authorized agent in an
effort to gain access to services and resources to which it is not
entitled. The masquerading agent may also pose as another
unauthorized agent in an effort to shift the blame for any
actions for which it does not want to be held accountable. A
masquerading agent may damage the trust the legitimate agent
has established in an agent community and its associated
reputation.

It could be a denial of service attack where mobile agents
launch attacks by consuming an excessive amount of the agent
platform's computing resources. This denial of service attacks
can be launched intentionally by running attack scripts to
exploit system vulnerabilities, or unintentionally through
programming errors. A rogue agent may carry malicious code
that is designed to disrupt the services offered by the agent
host, degrade the performance of the host, or extract
information for which it has no authorization to access.
Depending on the level of access, the agent may be able to
completely shut down or terminate the agent platform. It may
also occur in the form of unauthorized access where an agent
gains access and privilege to services and resources of the
host.

Access control mechanisms are used to prevent unauthorized
users or processes from accessing services and resources for
which they have not been granted permission and privileges as
specified by a security policy. Each agent visiting a platform
must be subject to the platform's security policy. Applying the
proper access control mechanisms requires the platform or
agent to first authenticate a mobile agent's identity before it is
instantiated on the platform. An agent that has access to a
platform and its services without having the proper
authorization can harm other agents and the platform itself.

3.4.2 Protecting the Mobile Agent from other Mobile
Agents
The agent-to-agent category represents the set of threats in
which agents exploit security weaknesses of other agents or
launch attacks against other agents. This could be in the form
of masquerading, unauthorized access, denial of service and
repudiation. Many agent platform components are also agents
themselves. These platform agents provide system-level
services such as directory services and inter-platform
communication services. Some agent platforms allow direct
inter-platform agent-to-agent communication, while others
require all incoming and outgoing messages to go through a
platform communication agent. Agent-to-agent
communication can take place directly between two agents or
may require the participation of the underlying platform and
the agent services it provides.
An agent may attempt to disguise its identity in an effort to deceive the agent with which it is communicating. An agent may also pose as a well-known vendor of goods and services, for example, and try to convince another unsuspecting agent to provide it with credit card numbers, bank account information, some form of digital cash, or other private information. Masquerading as another agent harms both the agent that is being deceived and the agent whose identity has been assumed, especially in agent societies where reputation is valued and used as a means to establish trust.

Agents can also launch denial of service attacks against other agents. This they can achieve by repeatedly sending messages to another agent, or spamming agents with messages, may place undue burden on the message handling routines of the recipient. Agents that are being spammed may choose to block messages from unauthorized agents, but even this task requires some processing by the agent or its communication proxy. If an agent is charged by the number of CPU cycles it consumes on a platform, spamming an agent may cause the spammed agent to have to pay a monetary cost in addition to a performance cost. Agent communication languages and conversation policies must ensure that a malicious agent does not engage another agent in an infinite conversation loop or engage the agent in elaborate conversations with the sole purpose of tying up the agent's resources. Malicious agents can also intentionally distribute false or useless information to prevent other agents from completing their tasks correctly or in a timely manner.

Repudiation occurs when an agent, participating in a transaction or communication later claims that the transaction or communication never took place. Whether the cause for repudiation is deliberate or accidental, repudiation can lead to serious problems that may not be easily resolved. An agent platform cannot prevent an agent from repudiating a transaction, but platforms can ensure the availability of sufficiently strong evidence to support the resolution of disagreements. This evidence may deter an agent that values its reputation and the level of trust others place in it, from falsely repudiating future transactions. Disagreements may arise not only when an agent falsely repudiates a transaction, but also because imperfect business processes may lead to different views of events. Since an agent may repudiate a transaction as the result of a misunderstanding, it is important that the agents and agent platforms involved in the transaction maintain records to help resolve any dispute.

If the agent platform has weak or no control mechanisms in place, an agent can directly interfere with another agent by invoking its public methods (e.g., attempt buffer overflow, reset to initial state, etc.), or by accessing and modifying the agent's data or code. Modification of an agent's code is a particularly insidious form of attack, since it can radically change the agent's behaviour (e.g., turning a trusted agent into malicious one). An agent may also gain information about other agents' activities by using platform services to eavesdrop on their communications.

3.4.3 Protecting the Mobile agent from the Host
This appears to be the most serious of the three fundamental security issues. No solution has been found for this problem yet, though effort has been made by some researchers to solve the problem and research effort is still being put into this area [1, 2, 21]. So far, partial solution is what has been achieved. Categories of threats in this area could be in the form of masquerading, denial of service attack, eavesdropping and alteration. Due to some of the issues discussed above, most hosts will not allow interaction with an agent for they may see them as worm (virus) especially given the fact that the operation of a worm is similar to that of agents and vice versa. Nonetheless, an attack of a malicious host on an agent by changing the agent’s behaviour or stealing secrets such as credit card information could pose a serious economic threat [6].

3.5 Social – Economic Factor
This is one area that should not be ruled out as hindrance to mobile agent adoption. Mobile agent technology was created to help remove some problems associated with distributed systems, but a situation where a technology will ensure reduction in economic value which is the quest for man's survival will definitely not be tolerated. Most sites carry out one form of advert or the other while some are heavily dependent on marketing advert. How often the sites are visited will go a long way affecting the wide spread of the products being show-cased on these sites. If mobile agents are indiscriminately allowed to interact with such sites in the place of human, then these adverts will of course become useless since adverts are made for human and such sites will loose revenue. This is one key aspect that should be put into serious consideration if there must be increase in the adoption of mobile agent technology.

4. DISCUSSION
Overall, the problem created by the choice of language of implementation of a mobile agent can easily be solved by understanding exactly what the agent should achieve and the characteristics the agent should possess. A mobile agent can exhibit a weak or a strong mobility. If a mobile agent is to achieve strong mobility then a language like Java is not apt for such an agent because Java has support for weak mobility. A language like Telescript with features that can enhance strong mobility should not be used if weak mobility is the developer’s quest. Besides, understanding the agency/platform where agents carry out its activities will help determine what language to use in order to ensure compatibility between agents and its agency.
The issue of coordination can easily be addressed if builders of agents are careful enough to develop agents with the mind that the agents will interact with one another. This will necessitate putting proper mechanisms in place to ensure coherence. The need for standardization arose as a result of differences in the philosophies of mobile agents’ vendors. With OMG and FIPA around, standards can be institutionalized to enforce some minimum requirements which will ensure commonality amongst different vendors.

Most of the issues responsible for low adoption of mobile agent contribute to security problems directly or indirectly. The degree of mobility is directly proportional to the level of security threat. So, if at the inception of the creation of the agent, one knows exactly what the agent is to do, the characteristics it should possess, then the developer will know what language can help realise such a mobile agent and as well be aware of the type of security mechanism to implement in the agent based on the chosen language. It is obvious that managing or proffering solution to other problems which contribute to security challenges will undoubtedly help mitigate security challenges and this in turn will help increase the adoption of the technology.

5. CONCLUSION

Mobile agent technology is a paradigm that involves the migration of an agent from one host to the other in performing a task on behalf of the user, suspending execution at one location and resuming at another location from where it stopped. Various benefit have been put forward for the use of mobile agent technology, these include; bandwidth reduction, reduction of network latency, autonomous execution, fault tolerance amongst others. This not withstanding, the adoption of the technology is dwindling due to barriers posed by the language of implementation, coordination, standardization, security, and social economic interest. Research geared towards addressing these mobile agent technology challenges is therefore of urgent importance to encouraging high adoption of mobile agent technology.

REFERENCES


