A New Paradigm for Mobile Agent Computing

R. B. PATEL and K. GARG
Department of Electronics & Computer Engineering
IIT Roorkee, Roorkee-247667,
Uttaranchal, INDIA,

Abstract: With the recent surge in Internet based users, the idea of Mobile Agent based computing systems has gained popularity. This is because agent based computing systems provide a very simple and powerful mechanism by which complex distributed computations can be constructed and performed over the Internet. This paper presents a new mobile agent paradigm which is an integration of the existing mobile agent paradigm with a stationary agent. We have developed a new mobile agent platform using this paradigm. A comparison study of the proposed paradigm with some existing mobile agent platforms with respect to reliability and computation power is also reported.

Keywords: - Agent Host, Mobile Agent, and Agent Submitter

1 Introduction
Mobile Agents (MAs) [20] can be seen as automated errand boys who work for the user. They can be regarded as an alternative or enhancement to the traditional client/server paradigm. While client/server technology relies on Remote Procedure Calls (RPC) [1] across a network, agents can migrate to the desired communication peer and take advantage of local interactions.

Many academic and commercial systems offer agent based computing platforms, such as Voyager from Object Space [9], [16], Aglets from IBM [3], Grasshopper from IKV [10], [12], Concordia from Mitsubishi [6] DAgent from Dartmouth [11]. Most of these systems are built for a special purpose or to solve a host of special problems. Currently, no system has generic architecture to support flexible and reusable components for modular system construction. Further none on these systems fail to protect the agent from malicious hosts. Most of today’s mobile agent systems (MASs) allow creation and cloning of agents, but effective mechanisms for termination of an agent and all clones are still missing, i.e., agents or clones are around in a network forever. Most systems lack efficient mechanisms for retransmission of an agent when a previously launched agent does not report after a fixed interval of time. When an agent accesses a host on the network, it uses its local resources. Therefore, there is a need for the resources of the host to be properly monitored and regulated. The agent should have capability to negotiate with the hosting environment for charges.

Most of these systems provide the fundamental features of agents, such as autonomy, intelligence, and mobility, but they all lack comprehensive support for resource utilization when failure arise at the next destination of the agent where they want to migrate and this failure remains for a long period of time. In this paper, we have modified the existing mobile agent paradigm (MAP) to a new one which improves the performance considerably.

The rest of the paper is organized as follows: Section 2 discusses the modified paradigm for agents. Section 3 gives a description of the developed system (PMADE: A Platform Mobile Agent Distribution and Execution [19]). Section 4 discusses the agent-naming scheme. Section 5 gives a brief illustration of agent architectures implemented in PMADE. Section 6 discusses the implementation and performance study of the developed paradigm. Section 7 illustrates the related work and Section 8 concludes the paper.

2 Mobile Agent Paradigm (MAP)
We propose a MAP {Fig.1}, which is an integration of the existing MAP [2] with a stationary agent. In this paradigm, each server requires a stationary agent, which is in charge of accepting the result generated by the agents after completion of their assigned task.

![Fig. 1. Mobile Agent Paradigm](image-url)

In the existing paradigm, when an agent finishes its task, but cannot return to its launcher due to some problems, it polls its launcher after regular intervals of time. Thus, some portion of the agent code is still executed. In other words, the agent uses CPU time to request for a connection with a remote agent launcher. Moreover, if several clones of this agent have been...
launched, all these will reside on remote servers [3] if they are unable to find the necessary connection to their launchers. Thus, a lot of resources, in terms of CPU, primary and secondary storage are wasted, when agents live after completion of their tasks and are unable to reach their destination.

The proposed paradigm provides great advantages, since when an agent finishes its assigned task, it submits its result to the local server in secure form and dies, thus optimizing the traffic on the network. Further, only one stationary agent is in charge of all the agents running on the local server. Still further, when several clones of an agent work on different systems and, after finishing their assigned task, travel on the network, traffic on the network is increased in the existing paradigm, but in the new paradigm, only results travel on the network.

When an agent is received at a server it may be cloned and forwarded to other active hosts. Where if required, clones of the clone are further created by the agent itself. One clone of the agent run on the host at which cloning is done. After finishing this assigned task, the agents submit the results to their respective hosts. The secondary hosts send results of cloned agents to the host from where the initial cloning was done. After receiving the results from all the clones, host combines these results and sends back to the agent launcher. In Fig. 2 we have shown three stages A, B and C. The agent is first received at stage A, the clone of the same is generated and forwarded to the hosts belonging to stage B. If required, further cloning is done by the agent itself and forwarded to stage C. This cloning can continue in further stages, if required.

**Fig. 2. Mobile Agent in Parallel & Distributed Computing**

Assume agent MA\textsubscript{1} of size 1.5 KB received at stage A, \(m\) clones are forwarded to stage B and \(n\) clones are forwarded to stage C. Assume that BR\textsubscript{1}, BR\textsubscript{2}, BR\textsubscript{3} ... BR\textsubscript{m} and CR\textsubscript{1}, CR\textsubscript{2}, CR\textsubscript{3} ... CR\textsubscript{n} KB of result data is generated by the cloned agents at stage B and C, respectively. The stage A generates result AR KB.

If we consider old MAP then we find that \(n\) (cloned agents are received from stage B) cloned agents running at stage C for MA\textsubscript{1}. These agents with results (agent code size *no. of clones KB) travel on the network after completion of assigned task to reach at stage B from where they have been received. After receiving agents from stage C at stage B, \(m\) cloned agents travel with results obtained at stage B as well as result received from stage C on the network to reach to stage A. Finally single agent (original) with all the results returns to its launcher. The same operation is done for other agents received at different hosts at different stages as shown in Fig. 2. In the next section we illustrate a system which implements this paradigm.

### 3 System Architecture

The main blocks of PMDAE are shown in Fig. 3. It is assumed that each host of the network hosts an Agent Host (AH: server), which is in charge of accepting and executing incoming agents and an Agent Submitter (AS), which is responsible for submitting the agent on behalf of the user to the AH.

When a user wants to perform a task, he/she submits the agent, designed to perform that task, to the AS on the user system. The AS then tries to establish a connection with the specified AH, where the user is registered. If the connection is established, the AS submits the agent to the AH and then goes offline. The AH examines the nature of the agent and if required, makes its clone and forwards it to other AHs, which are active in the network. It then goes on to execute one clone.

The execution of the agent depends on its nature and state. The agent can be transferred from one AH to another whenever required. On completion of execution, the agent submits its results to the AH, which in turn stores the results, until the base host (BH) (Note that PMDAE installed on the router of a network is working as gateway to PMDAE Agents)/AS receives them for the user.

#### 3.1 Agent Submitter (AS)

The AS plays a crucial role in formulating and dispatching the agent to the AH. It acts as an interface between the user and the AH, as shown in Fig. 3. One of its primary jobs is to attach a signature [4] to the invoked agent. It retrieves the static IP address of the host on which it is running and binds it to the agent signature. This enables the agent in the itinerary to send the reply back to the client (from where the agent was launched). The user’s name in the agent signature is the logon name with which user logs on to the AH [4]. In the agent signature, an important field is the name of the agent. The submitter has to ensure that it assigns a unique name (see section 4) to each agent that it submits to the AH. The version field of the agent signature is used to keep track of the revisions made in its code.

The AS is also responsible for receiving replies from the AH for requests from the user. To achieve this, the AS also keeps track of and maintains a profile of all the agents that it submits to the AH from the user’s system. Thus, this architecture allows the users to go
offline after submitting their agents and receive the results whenever he/she reconnects to the network and requires the results.

3.2 Agent Host (AH)
The AH is the key component of PMADE. It consists of the manager modules and the Host Driver. The Host Driver is the basic utility module, which lies below the manager modules and is responsible for driving the AH, by ensuring proper co-ordination between the various managers and making them work in tandem.

![Diagram of Agent Host](image)

Fig. 3. Architecture of PMADE

The Service Bridge provides hosting facility to a carrier agent. The carrier agent regularly monitors the requests issued by the AH. If it receives a request to transfer an agent to another AH, it takes immediate action. It exploits the services of HTTP protocol for transferring an agent. Various manager modules help to perform various functions like transfer, execution, communication, etc. of agents. The managers of PMADE are grouped in four categories:

1. Communication
2. State Management
3. Persistence
4. Security

A detailed illustration of the different components of developed paradigm is published in [19].

4 Agent Naming Scheme
In designing of the MA naming scheme, the main question to address is what the Agent Name has to represent and how it is chosen and used? A good way to select the name is to make it reflect the functionality provided or the service offered by the agent, as is the case of the Internet, for example, where the convention for server names is to use a form of the type service.organization.domain (e.g., www.iitr.ernet.in).

The agent-naming scheme developed in PMADE is based on MASIF (Mobile Agent System Interoperability Facility), the standardization OMG (Object Management Group) [6], [5] proposal for MASs. The entire environment is subdivided as follows:

A Place is the environment where agents perform execution. Each place resides in a separate host and consists of the AH and resource variables that the agent access during execution.

A Region is a homogeneous pool of places, each capable of hosting agents of a particular agent service.

In the distributed environment outlined, an agent is created in a place and travels from place to place across AH and regions. An agent is thus characterized by information on two locations:

- The region, AH and place where the agent is born
- The region, AH and place where the agent is currently located.

In the following, we will refer to a global location identifier (GLI), unique in the distributed environment, consisting of three local identifiers related to the place, the AH and the region, respectively. This definition is quite general and can be adapted for any MAS. Given this, we will represent the agent’s birth GLI and its current GLI, respectively, with the symbols BGLI and CGLI. In PMADE an agent name $an$ has the following form:

$$an : = \text{“agent:” localname “@” BGLI.region},$$

where BGLI.region is the name of the agent’s region of birth and localname is its name chosen by the agent programmer.

Two or more regions of the same name, or two or more agents, generated in the same region, with the same name are not allowed. The uniqueness of the region name can be guaranteed by an authority which assigns a name to a region when the latter is created (generally, this is a patchy activity). On the other hand, in order to ensure the uniqueness of localname, a binding protocol is needed to register the new agent and check that its name is unique in that region. This naming scheme possesses both location independence and selectability properties, and possesses transparency partially since the presence of the name of the birth region imposes a kind of network related information.

In the literature, it is found that only a few MASs implement the concept of region. Many MASs, such as Aglets, Voyager, Concordia, use only the concept of agency (with no subdivision in places), identified by the TCP port number to which the AH is bound and the host name. However, implementing the above naming
scheme into a MAS of this kind is still possible, by using an implicit or explicit region-binding scheme. Explicit binding can be done by manually assigning a region of ownership to each MAS running in the system (and implementing the suitable region management protocols). Implicit binding can be done on the basis of the network site name: usually, the site name has a hierarchical structure which imposes an implicit aggregation of the sites whose names belong to the same hierarchy level.

It should be noted that in PMADE, the naming scheme is quite similar to the one used in assigning e-mail addresses in the Internet (e.g., agent: task@iitr.ernet.in). This is deliberate, since an agent who usually does some work on behalf of the user, can represent the user in the network, and has a human-like behavior. Thus, a naming scheme similar to the one used to address a user on the network seems a good choice. In addition, by using an Internet-style representation, DNS protocols can help us in the location handling process.

In some MASs, such as Voyager and Grasshopper, the couple (host, port) points to a reference AH from which the location finding process starts. As this information does not vary during an agent’s life, in this case the naming scheme meets the location independence property. In other MASs, such as Aglets and D’Agents, the couple (host, port) has to point to the agent’s current location, which has to be known in advance; in this case, the naming scheme does not meet the location independence property.

### Table 1. Naming Schemes comparison

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Example</th>
<th>Transparency</th>
<th>Location Independent</th>
<th>Selectability</th>
<th>Used in MAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current place + id</td>
<td>LocalHost:4342</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Agent</td>
</tr>
<tr>
<td>Current place + name</td>
<td>LocalHost:1234AgentUser</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Agent:Task, D’Agents, Grasshopper</td>
</tr>
<tr>
<td>Federation-level + name</td>
<td>Federation:5001AgentName</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Voyager, Grasshopper</td>
</tr>
<tr>
<td>Name + Hosting region</td>
<td>AgentName@Region01</td>
<td>Port</td>
<td>Yes</td>
<td>Yes</td>
<td>PMADE</td>
</tr>
</tbody>
</table>

As for the agent-identifier, the majority of MASs allow the programmer to autonomously choose an alphanumeric name, while a few (including Aglets) use a numeric identifier, which is automatically assigned by the MAS, when the agent is spawned. Only in the former case does the naming scheme possess the selectability property. Table 1 summarizes the comparisons between the naming schemes available in other MASs PMADE.

In PMADE the Network Address Manager works like an authority for assigning region names. It manages uniqueness of region names, and also maintains information about an AH and its region. When AH starts, it can be a member of an existing region or can start a new region. In each region, at one AH, we maintain a Region Agent Database (RAD), which manages all information about the agents which are created in that region or transit through it. At each AH we are maintaining Site Agent Database (SAD), which manages all information about the agents which are created in that region or transited through it. When a new agent is created, the user can assign a name to agent by registering in RAD of birth region, it also have to register its information in SAD of local AH.

### 5 Agent Architecture

A novel aspect introduced in PMADE is the extendibility provided by the agent architecture. Instead of focusing on a specific type of agent, we have implemented different agent architectures. Even mobility is a feature of only some agent architectures. Moreover, our design allows several types of agents coexisting at the same time in a heterogeneous environment. The architecture of an agent is an adaptable model that determines the different parts in which it is divided and the combination of security, integrity or other mechanisms included in it. The implementation of agent management methods are included in the agent itself, if it is designed for implicit itinerary, since they are part of the agent architecture. This allows delegating specific mobility functions to the agent, freeing theAH from this responsibility. Otherwise, an agent is managed by the AH. Using this implementation, PMADE becomes more generic.

All built-in MA architectures in PMADE can be categories into two types, AH managed and agent managed. The agents share the same basic aspects, such as the internal architecture and migration mechanisms. In our implementation, a MA consists of agent code, agent container, results and itinerary type. Agent code is the set of instructions describing the task of the agent. Agent container is an information storage area that can be used by the agent at any time for reading and writing and goes with it all the time. Agent container is also used to carry the Task objects (methods), when specific object for specific host concept is used to implement an agent [7]. Results are the output generated by the agent. Itinerary type provides two options to the agent programmer as shown below:

#### Option-A: External (Explicit) Itinerary

In each itineraries agent code is split into several pieces. Each piece of code and itinerary are secured using the protocols in [7].

1. **Serial Itinerary-** in this itinerary an agent is controlled and managed by the AH. When it finishes its task, it transfers its control to the AH and is transferred to the next AH in the itinerary.
2. **Virtual Serial Itinerary-** in this itinerary an agent is controlled and managed by the AH on which it is initially submitted.
3. **Parallel Itinerary-** in this itinerary agent is also controlled and managed by the AH on which it is initially submitted.
The main code (Common code) is executed in all hosts in the serial itinerary, but in virtual serial and parallel itineraries, common code is only executed at the AH where an agent is initially submitted. As many task objects as hosts are in the itinerary, each one to be executed in a particular host. This feature makes PMADE very useful in applications where execution is context dependent. It should be noted that common code, which is needed to start the execution of task object at every host, is embedded in AH which can be employed by the specific itinerary controller routine to execute specific method at an AH. Thus, in the above itineraries, some network load is reduced.

**Option-B: Internal (Implicit) Itinerary**

The implementation of agent management methods is included in the agent itself, since they are part of the agent architecture. This allows delegating specific mobility functions to the agent, freeing the AH from this responsibility. There are two cases shown below:

1. **Agent with pure Implicit Itinerary**

Here an MA does not contain a separate itinerary, but it is merged in the agent code. A simple method call allows the agent to migrate to other AH. The code of such a MA is \( MA_i = \text{Control Code}, \text{Task Code}, \text{Agent Container}, \text{Dynamic Data}, \text{Static Data} \).

The same code is executed by every AH. Therefore, there is no need to keep the agent code secret. Third parties cannot spy on the agent code if secure migration is done. When secure communication is not available, we need to implement some security technique so that the agent can be safely transferred on the network. For securing the agent code during agent migration, we have designed and implemented the following algorithm for implementing the implicit itinerary in the agent code.

If (Task is not completed)

Take the help of directory services for identifying the locations where the required services may exist.

1. Read the public key of the next host where the agent has next to migrate, from the key store of current host,
2. generates a key pair by using the public key of the host where it has to next migrate,
3. add generated public key to its Container,
4. Sign and seal the agent’s Task code with the public key of the next host in the itinerary,
5. add the Task code, generated result and intermediate states to agent container and create new instance of agent object for the next host,
6. Add this sealed object to migration container,
7. call the migration routine and migrate, set the status flag.

If (status = true) Collects garbage
Else try next host where the required services may exists, repeat step 1 – 7.

} else{

read private key of the current host,

Sign and seal the generated result so far from the itinerary.
Add this sealed object to container.
Call migration routine and migrate, set the status flag.
If (status = true) Collect the garbage
Else Forward this sealed object to Agent Reply Manager of the host.

2. **Agent in Implicit Itinerary with Explicit Property**

Here agent code is split into several pieces, each piece of code and itinerary, secured using the protocols in [7]. There is a main code (Common Code) that executes in all hosts and as many task objects as hosts are in the itinerary, each one to be executed on a particular host. This feature also makes PMADE very useful in applications where execution is context dependent. The agent changes after every migration. This dynamic aspect of the agent allows several security mechanisms to be applied. In this architecture, the agent has structure \( MA_i = \text{Control Code}, \text{Common Code}, \text{Agent Container}, \text{Static Data}, \text{Results}, S_e(\text{H (Control Code, Common Code)}), \text{Host}, \text{Object RH}, \text{Host RH} | \text{NULL} \)

Host, is the host where the agent next migrates. The agent that is sent to the next hop of the itinerary \( (MA_{i+1}) \) has the same structure. The last host is identified because it has a NULL for next agent. Common code is executed by all AHs (hosts) when the agent migrates and before the specific task object. Programming is simplified by using this common code. This code does not depend on the AH and is included only once.

This architecture allows several protection mechanisms in addition to the mechanism presented in pure implicit itinerary architecture to protect the agent code and data on an insecure communication channel. Further, code and itinerary are also protected for agents using this architecture. The idea is to take advantage of distributed sealed object and re-router host [7] in the agent itinerary to make the agent’s code secure and its itinerary secure and robust. Thus, only a portion of code is available to AH for execution. The following is structure of the agent when using agent container, agent code, and itinerary protection.

\( MA_i = \text{Control Code}, \text{Common Code}, \text{Agent Container}, \text{Static Data}, \text{Results}, S_e(\text{H (Control Code, Agent Container)}), \text{Itinerary}, \text{Itinerary}_{i} = E_i(\text{Agent Container, Host, Host RH}) \) | NULL

where \( E_i() \) is an encryption function using public key of the Host_i.

It should be noted that Agent container contains the list of objects, each of which is executed at particular host. This container also contains a special object that is used to re-inject the agent in the itinerary, when the next host in the itinerary is unreachable.
6 Implementation & Performance Study
We have compared the performance of PMADE (version 1.0), with that of Aglet (version 2.0.1) and Grasshopper (version 2.2.3b) systems. We conducted 4 experiments on a setup of three networks with net IDs 192.168.0.X, 192.168.110.X, and 192.168.111.X connected in a large Intranet. All hosts in the Intranet need not have the same configuration. For e.g., our hosts are P-I 166 MHz, 64 MB RAM, Windows 98, P-II 233 MHz 64 MB RAM, Windows 98, P-III 500 MHz, 64 MB RAM Windows 2000/Linux, P-IV 1.5 GHz, 128 MB RAM and P-IV 2 GHz 256 MB RAM, Windows 2000/Windows XP/Linux, respectively. All are equipped with j2sdk1.4.1_02 middleware from Sun Microsystems and run the PMADE system. We have carried out 3000 trials in one week at different times and at different network load. The results shown in Figs. 4-7 are average. In each test, we varied the size of the payload carried by the agent and set it to 1 KB, 10 KB, 100 KB, 1 MB, 3.62 MB, 10 MB, 12.5 MB, 15 MB, 20 MB, 30 MB, 60 MB and 75 MB. The size of the bare agent without payload was about 8 KB.

Performance indices chosen are agent migration time, and itinerary time with variable size of data and CPU overhead with varying number of agents. We started agent platforms in two computers 192.168.0.3 (P-IV 1.5 GHz, 128 MB RAM, Windows 2000) and 192.168.0.4 (P-IV 2 GHz 256 MB RAM, Windows XP), with 192.168.0.3 as AS host and 192.168.0.4 as destination host. AS dispatches an agent (namely Distribution Agent) to 192.168.0.4 and receives it back after completion of its assigned task. The agent carries some data during execution. It copies this data into the secondary storage of the destination host. While coming back, it reports the status, i.e., whether it has delivered the carried information to the destination host or not. The results are shown in Figs. 4-5.

We have evaluated the performance of Aglet and PMADE, for the distribution of Norton Antivirus definition on a cluster of 100 machines. Results are shown in Fig. 6. It can be seen that the Aglet’s performance continuously deteriorates as the number of hosts and data size increases as shown in Fig. 4 & Fig. 5. The cloning power of Aglet and Grasshopper also deteriorates with increase in size of data carried by the agent. The Aglet system shows an error message “out of memory” at the data size of 12.5 MB. Grasshopper does the same and flashes the message “registration error” at 15 MB. The agent dispatcher maintains multiple copies of the agent and clone in the system when it is ready to transfer them across the network and poor garbage collection in both Aglets and Grasshopper affects the system performance. However, PMADE works correctly up to data size 72.5 MB without any performance degradation and flashes the message “connection refused”, for greater size data.

It is also observed that, in the Aglet system, if an agent enters with a data size 100 KB, with 125 agents already running in the system with the same data size, then system flashes “out of memory” making all the running agent exiled. When the system will get recovered, they will not start. But, in case of Grasshopper and PMADE, all the agent will be restarted after recovering of system from the fault.
7 Related Work
Many MASts have been developed in recent years. Some of them are briefly discussed below:

Telescript [14], developed by General Magic, includes an object-oriented, type-safe language for agent programming. Telescript has significant support for security, including an access control mechanism similar to capabilities. Each agent and place has an associated authority, which is the principal responsible for it. A place can query an incoming agent’s authority, and potentially deny entry to the agent or restrict its access rights. The agent is issued a permit, which encodes its access rights, resource consumption quotas, etc. The system terminates agents that exceed their quotas, and raises exceptions when they attempt unauthorized operations. Telescript was not commercially successful, primarily because it required programmers to learn a completely new language.

Tromsø and Cornell Mobile Agents (TACOMA) [13] was developed at University of Tromsø, Norway & Cornell University USA, with focus of fault tolerance scheduling and management, security & accounting of agents. Agents are written in Tcl, although they can technically carry scripts written in other languages too.

Aglets were developed by IBM Tokyo Research Lab. Aglets are serialized Java objects that execute on Aglets Workbench [3]. A developer can use the classes and methods defined in Java Aglet API [3] for aglet creation and manipulation. The mobility of the aglet is achieved by the serialization and dynamic class loading techniques of Java. An aglet serializes itself and dispatches to another Aglet Workbench, where it is loaded (executed) by the class loader.

Concordia, developed by Mitsubishi Electric Information Technology Center America [6], provides a Concordia Server that executes on top of the Java Virtual Machine as the agent platform. Concordia’s security model provides support for three types of protection: agent storage protection, agent transmission protection and server resource protection.

D’Agent, formerly named Agent Tcl, is one of earliest MASts. It was developed in Dartmouth College [18], [11]. Unlike Java based MASts, D’Agent uses Tcl interpreter as the agent platform. D’Agent also supports several kinds of programming languages, including Tcl, Java, and Scheme.

The Secure and Open Mobile Agent (SOMA) [15] developed at the University of Bologna, is another MAS implemented in Java. What makes SOMA unique from other Java based MASts is that it interoperates with CORBA [9], so that it can be integrated into CORBA-compliant environments. A SOMA agent executes in an environment called SOMA place, which represents physical machines, and the SOMA places can be grouped into domains that represent LANs. Places and domains provide two layers of abstraction that represent the Internet.

Voyager [16] is a pure Java agent-enhanced Object Request Broker developed by ObjectSpace Inc. It provides for creation of both autonomous agents and objects. Voyager agents roam a network and continue to execute as they move. Voyager can remotely construct and communicate with any Java class, even third party libraries, without source. It allows seamless support for object mobility.

GMD FOKUS and IKV++ GmbH have developed Grasshopper [10], [12], which is an agent development and runtime platform, built on top of a distributed processing environment. This achieves an integration of the traditional client/server paradigm and MA technology. Most importantly, Grasshopper has been designed in conformance with the first agent industry standard, namely the Object Management Group’s Mobile Agent System Interoperability Facility [8].

Ajanta is an agent programming system being developed at the University of Minnesota [17]. It allows agents written in Java to securely migrate from machine to machine on the Internet. The main focus of the Ajanta design is on mechanisms for secure and robust execution of agents in open systems. In Ajanta, the MAP is based on the generic concept of a network mobile object. Its security mechanisms are designed based on Java’s security model.

The true strength of MASts is that they are a uniform paradigm for distributed applications. Thus, all the existing systems are intended for general applications, differing only in their language, migration, security, fault tolerant and support services. PMADE distinguishes itself by providing a generic system which support multiple agent architectures, which can be chosen by the application developer according to the need of an application. Other major difference in paradigm implementation is a stationary agent that watches every agent running on a host and removes those agents from the RAM when they wait for the next event to happen.

8 Conclusion
In this paper, we have presented a novel MAP. Many security features are implemented in agent architectures, whilst others are provided as internal services. PMADE allows the agent programmer to use pure explicit, pure implicit and implicit with explicit property of agent architectures for their application development. PMADE maintains overall integrity of an agent on the network during its transport. It also supports MASIF standard agent naming services.

There are several advantages in using a single stationary agent for forwarding results of all the agents running on a host. In this technique, when the list of active hosts is exhausted and the AS is inactive, the agent is placed on the secondary storage. The ARM is now in charge of this agent. The ARM tries (for this uses itinerary controller of Agent Manager) to find the
status of the next host in the itinerary or if in the mean time if no active host is available and AS becomes active agent’s result is sent to AS. Other systems do not have this property and agents after completion of assigned task to them at a host, if there is no further reachable host, they wait on the current host. This technique creates CPU, RAM and secondary storage overhead. In Aglets, an agent remains in exile when no link is found to its launching host. Thus, the advantages of our approach are:

1. It saves network bandwidth, because only the agent’s result travels on the network and not the whole agent. When there are several clones of an agent running, saving in bandwidth is quite substantial.
2. It also saves secondary as well as primary storage at the remote AH.
3. From the security point of view, agents are required to report results to the AH in a secure manner, so that further tampering of the result is not possible by the AH. In PMADE, the agent provides the result in encrypted form to the AH.
4. If an agent is running on a machine and the machine is disconnected from the network, the agent’s results remain in exile on this machine for a fixed period of time. When the machine is connected again, the result of the pending agent is sent to its BH/AS. But if this duration is large, the AH removes the entries of the pending agents and the result is lost.
5. Except for the Signature and Agent Managers, all manager modules are mobile.
6. It is possible to add new modules within PMADE without affecting the existing modules.

References