

SOFTWARE DISTRIBUTION USING MOBILE AGENTS

A DISSERTATION

*Submitted in partial fulfilment of the
requirements for the award of the degree*

of

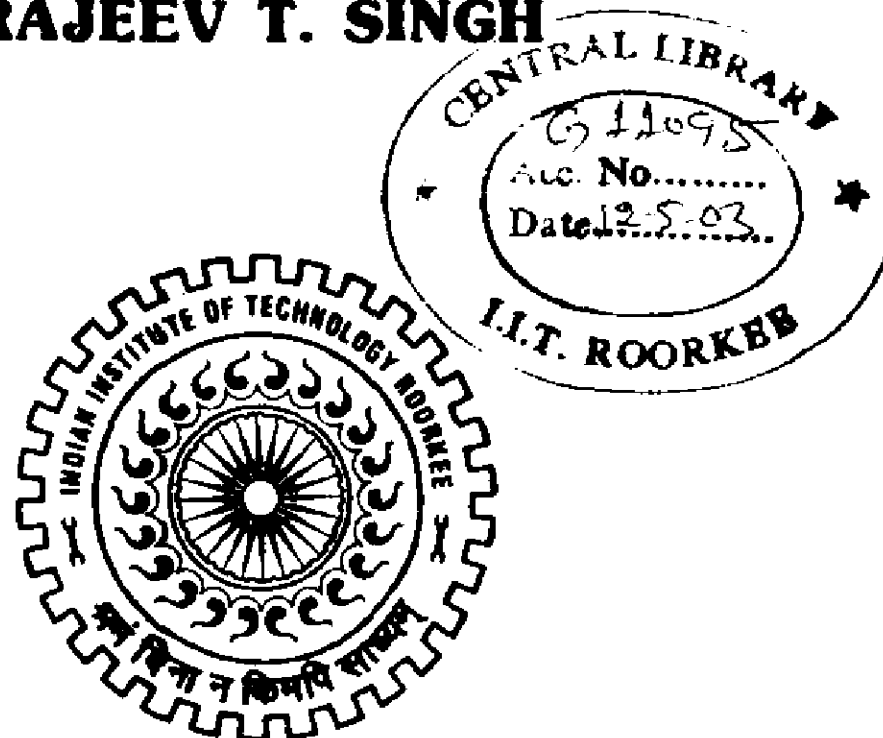
MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

RAJEEV T. SINGH



**ER & DCI
NOIDA**

**IIT Roorkee-ER&DCI, Noida
C-56/1, "Anusandhan Bhawan"**

Sector 62, Noida-201 307

FEBRUARY, 2003

CANDIDATE'S DECLARATION

This is to certify that the work, which is being presented in this dissertation, entitled "SOFTWARE DISTRIBUTION USING MOBILE AGENTS", in partial fulfillment of the requirements for the award of the degree of Master of Technology in Information Technology submitted in IIT, Roorkee – ER&DCI Campus, Noida, is an authentic record of my own work carried out from August 2002 to February 2003, under the supervision of Mr. P.N. GOSWAMI, Director, R&D, Electronics Research and Development Centre of India, Noida.

I have not submitted the matter embodied in this dissertation for the award of any other degree.

Date: 21.02.2003

Place: Noida



(Rajeev T. Singh)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 21/02/2003.

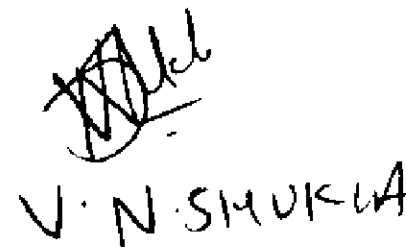
Place: Noida



(Mr. P.N. Goswami)

Director(R&D),

ER&DCI, Noida



V. N. SHUKLA

ACKNOWLEDGEMENT

I would like to thank Prof. Prem Vrath, Director, IIT-Roorkee, Mr. R.K. Verma, Executive Director, ER&DCI, Noida, Dr. A.K. Awasthi, Dean, Post Graduate Studies and Research and Dr. R.P. Agarwal, Course Coordinator for giving me this valuable opportunity to pursue this work.

I would like to thank my guide, Mr. P. N. Goswami, Director, R&D, ER&DCI, Noida for his patience and encouragement. He has been an invaluable source of support and guidance all through this dissertation.

I am grateful to Mr. V. N. Shukla, Course Coordinator for providing the best of facilities to carry out the dissertation. My sincere thanks are due to Dr. P.R. Gupta for the encouragement and valuable suggestions she provided me with during the course of my work. I am also grateful to Mr. Munish Kumar for the cooperation extended by him in the successful completion of this report. I would also like to thank Mr. R. B. Patel, Research Scholar, IIT, Roorkee for his timely help and constant guidance.

Most of all I would like to thank my family. My parents provided me a perfect environment for my studies and supported me throughout. Finally, I would like to extend my gratitude to all those persons who directly or indirectly helped me in the process and contributed towards this work.



(Rajeev T. Singh)

019033

CONTENTS

CANDIDATE'S DECLARATION	(i)
ACKNOWLEDGEMENT	(ii)
ABSTRACT	1
1. INTRODUCTION	3
1.1 Overview	3
1.2 Objective of Dissertation	3
1.3 Scope of the Work	4
1.4 Organization of Report	4
2. LITERATURE SURVEY	5
2.1 Software Distribution	5
2.2 Introduction to Mobile Agents	7
2.2.1 Mobile Agents	7
2.2.2 Agent Server	8
2.2.3 Mobility	8
2.2.4 Agent Architecture vs. Client/Server Architecture	8
2.2.5 Advantages of Mobile Agents	9
2.3 Mobile Agent Framework	11
2.4 Java and Mobile Agents	12
2.5 Existing Mobile Agent Systems	13
2.5.1 Telescript	13
2.5.2 Agent Tcl	14
2.5.3 Voyager	14
2.5.4 Concordia	14
2.6 Aglets	15
2.6.1 Where to use Aglets?	15
3. AGLET SPECIFICATION	19
3.1 Architecture Overview	19
3.1.1 Aglets Runtime Layer	20
3.1.2 Communication Layer	21

3.2 Aglet API Overview	23
3.2.1 com.ibm.aglet.Aglet	23
3.2.1 com.ibm.aglet.AgletProxy	24
3.2.1 com.ibm.aglet.AgletContext	24
3.2.1 com.ibm.aglet.Message	25
3.3 Aglet Object and its Life Cycle	25
3.4 Security in Aglets	27
4. SYSTEM DESIGN AND IMPLEMENTATION	29
4.1 IBM's Aglet Software Development Kit	29
4.1.1 Tahiti server	29
4.2 Design of Mobile Agent on Aglet Platform	30
4.3 Agents	31
4.3.1 System Property Agent	31
4.3.2 Listing Agent	32
4.3.3 Distribution Agent	32
4.3.4 Delete Agent	33
5. RESULTS AND DISCUSSION	35
5.1 Tahiti Interface and Creation of Aglet	35
5.1.1 Creation of Aglet	35
5.2 Software Distribution Agents	37
5.2.1 System Property Agent	37
5.2.2 Listing Agent	39
5.2.3 Distribution Agent	41
5.2.4 Delete Agent	43
6. CONCLUSION	45
REFERENCES	47
APPENDIX – A Tahiti Menu Interface	

ABSTRACT

This project is an attempt to simplify the task of software distribution by using mobile agents. The aim of this work is to provide the network administrator with a tool, which saves him from the tedious routine of physically going to each and every node of the network in a typical organization scenario, in the process of distributing the required software on the nodes. Mobile agents are simply software agents that are not bound to the system where they begin execution. They have the ability to transport themselves from one system to another remote location to complete their task there. Software Distribution is one of the area in which mobile agents can be used effectively. A Distribution agent can carry software across the network reducing the workload on the network administrator. The project uses Aglets SDK as the mobile agent platform developed by IBM's Tokyo Research Lab for the development of software distribution aglet. Here the main focus has been on carrying software to remote location and initiating the installation procedure. Several clones of this agent can be created and software can be attached to these clones and sent to different system across the network.

INTRODUCTION

1.1 Overview

Network management is gaining importance due to the explosive growth of the size of computer networks. The network manager is faced with an increasingly complex job of managing a typical network. As information and its access becomes more dispersed and diverse the challenge is to discover new technologies to assist in this information management. The use of agents, in particular mobile agents is one such emerging technology that may assist in this information management process.

Mobile agent technology is nowadays one of the most active research topics in computer science. It is now proven that many areas could benefit from mobile agents. Many papers like Mobile Objects and Mobile Agents: The Future of Distributed Computing? [1] investigated on mobile agents' concept and theories. Most of the current research on mobile agents has two general goals: reduction of network traffic and asynchronous interaction. The research is driven by the fact that mobile agents will be soon developed at larger scale, for commercial application. Kotz and Gray [2] predict that, within a few years, nearly all major Internet sites will be capable of hosting and willing to host some form of mobile code or mobile agents. But before such a deployment, some issues need to be investigated and some problems need to be fixed. Security of mobile agent systems and interoperability are surely the main concerns for commercial development of this technology.

1.2 Objective of the Dissertation

Mobile agents provide an easy way to transport code and install packages automatically. They enable applications to distribute themselves among various computers on which they must execute. If package is available on a server, the application can expand to encompass any number of client computers.

The network administrator always faces a complex task of maintaining and upgrading systems in any organization. Whenever a new upgrade or a patch arrives the administrator has to physically move to each and every node in the network and run the

package. The purpose of this dissertation is to explore possibility of a system using which the administrator can transport the package to remote nodes and run them. Mobile agents provide the best answer for the solution of this problem.

1.3 Scope of the Work

This dissertation uses IBM's Aglet platform for developing mobile agents for software distribution. Software Distribution using Mobile Agents will provide analysis, notification, distribution, and installation of software and its updates to network-based computer systems. Here the main focus has been on installation (carrying software to remote computers and initiating installation procedure). It also contains Listing agent, System property agent and the Delete agent in support of the main Distribution agent. Following are the scenarios in which a agent can be used

- A new program can be installed on a company's computers.
- A software company installs a software package using this agent.
- A software company distributes the latest patch using the Internet.
- An error occurs in a machine. The manufacturer sends a mobile agent to examine the problem.

Advantages of using this approach are

- No special client needed, just the agent server.
- The agent server could be used to perform other tasks as well.
- It is easy to use always the newest version of a maintenance program.

1.4 Organization of Report

This report contains six chapters. Chapter 1 gives the overview and discusses the objective and scope of dissertation. Chapter 2 gives an introduction to software distribution and mobile agents. It discusses all the relevant issues regarding mobile agents. Chapter 3 gives a description of the aglet platform. Chapter 4 deals with design of agents for software distribution. Chapter 5 describes the user interfaces of the agents. This chapter also discusses the working of the agents. Chapter 6 concludes the thesis.

LITERATURE SURVEY

2.1 Software Distribution

The concept software distribution is a part of software deployment, which refers to all the activities that make software systems available for use. It comprehends the process and activities related to the release, installation, activation, deactivation, update, removal and retirement of software components in a set of hosts [3].

Once deployed, a software system is available for use on a customer site. A site may be a host or set of hosts that uses a set of resources. A **software system** is a coherent collection of artifacts, such as executable files, source code, data files and documentation. A **resource** is anything needed to enable the use of software system at a site, for example, and IP port, memory, disk space and other system. A **software producer** is a company or site that creates and deploy new releases of the software to be installed. The software consumer is the host in which the software needs to be deployed to.

The deployment process consists of several inter-related activities that can be executed on various hosts such as the producer, consumer or both. The software deployment process is a composition of following phases.

- **Release.**

Release is the activity that interfaces between the software development and its deployment. It is performed in the producer side and encompasses all the operations needed to prepare a system for assembly and transfer to the consumer site. It collects and specifies all information necessary to carry the other activities of the deployment process. In this phase, all components necessary to the application are collected and organized, in order to be transferred to the consumer sites. Such information comprises the components, documentation, its installation procedures, dependencies and management properties.

- **Installation.**

Installation covers the transfer of the application component from the producer site to the consumer site, followed by their configuration. It prepares the system to be activated.

- **Activation.**

Activation is the activity of running the installed application in the customer site. For complex systems it might require the initialization of other services and process.

- **Deactivation.**

Deactivation the inverse of activation activity. It performs the shut down of the running application. It is also required before other deployment activities can take place, for example, during update operations.

- **Update.**

Update is a special case of installation. It represents the partial or total transfer of new component versions, in order to replace components of an existing installation.

- **Adaptation.**

Like the update activity, the adaptation involves the modification of a software system that has been previously installed. Adaptation differs from update in that the update activity is initiated by remote events, such as software producer releasing a new component version, whereas adaptations are initiated by local events, such as a change in the environment of the consumer site. For example, the installation of a new graphic card may require the system to adapt to its new characteristics.

- **De-installation.**

This activity consists in the removal (undo) of the application components from the system. As a result, the remove process must inspect the current state of the consumer site. This procedure must not affect other installed systems, and dependencies check must be performed in order to keep components that are shared with other applications.

- **Retirement or Derelease.**

This phase consists in discontinuing the support for an application by the software producer. It usually requires that the withdrawal of the software by the producer be advertised to all known consumers of the system. It does not directly affect the consumers, who can continue to use the software. In summary, the producer site is responsible for the release and retirement of the software, while the consumer site performs the activation, deactivation and adaptation of the software.

2.2 Introduction to Mobile Agents

2.2.1 Mobile Agents

A mobile agent is an agent that has one more characteristic: its code is mobile. While a stationary agent executes its code on the same host all its lifetime, a mobile agent has the ability to transport and execute itself over a network, in a heterogeneous environment.

By code mobility, it is meant that the mobile agent not only transfers its code, but also it's being: code, data and state. It is possible for it to begin an operation on one host and continue it on another, while updating its data after the visit of each host.

The agent performs its job wherever and whenever appropriate and is not restricted to be co-located with its client. Although a mobile agent is essentially an executing process, the governing factor that distinguishes it from a normal process is that not all of its instructions have to be executed on the same node or even within the same network locale. With these abilities come certain benefits such as reduction in network traffic, protection against network latency, protocol independence, parallel computations, and adaptation to the working environment, and robustness. Mobile agents comprises of four main elements

- The agent itself with its properties and attributes.
- The place where the agent executes, or an environment where they can perform execution.
- The behavior of the agent, which comprises of creation, disposal, and transfer.
- Agent communication

2.2.2 Agent Server

An agent server is also known as the agent execution environment. An agent server controls agents: it creates, executes, transfers and terminates. It provides some services such as inter-agent communication.

2.2.3 Mobility

A feature of mobile agent systems is their ability to move mobile agent from one place to another. There are two ways to support mobility of a mobile agent: weak and strong migration.

In strong migration, the agent is transferred with its code, data and its complete state: it allows the mobile agent to be executed exactly from where it was left before migrating: it resumes itself.

Weak migration consists of transferring the code and data.

2.2.4 Agent architecture vs. client/server architecture

In the traditional client server architecture, all connections to server are initiated from the client. The advantage is that these connections can be managed in parallel (Fig.2.2.4.1). With an agent architecture (Fig.2.2.4.2) the network is dynamic, and the client need not know the structure of the network. The fact that agents communicate with a high-level communication language and work over a logical network makes them more reliable as the logical network which is dynamic will adapt itself to the current conditions and be less affected by network failure.

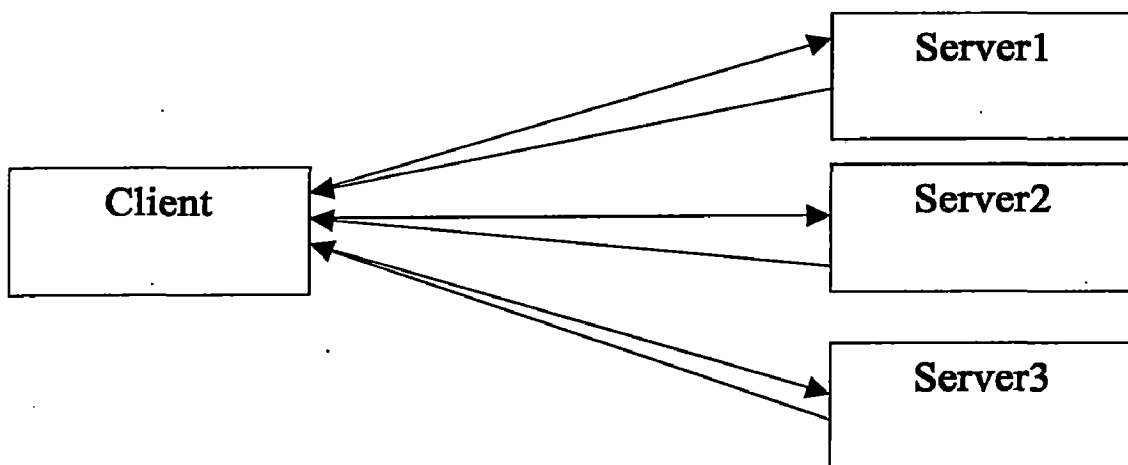


Fig.2.2.4.1: Traditional Client/Server Communication.

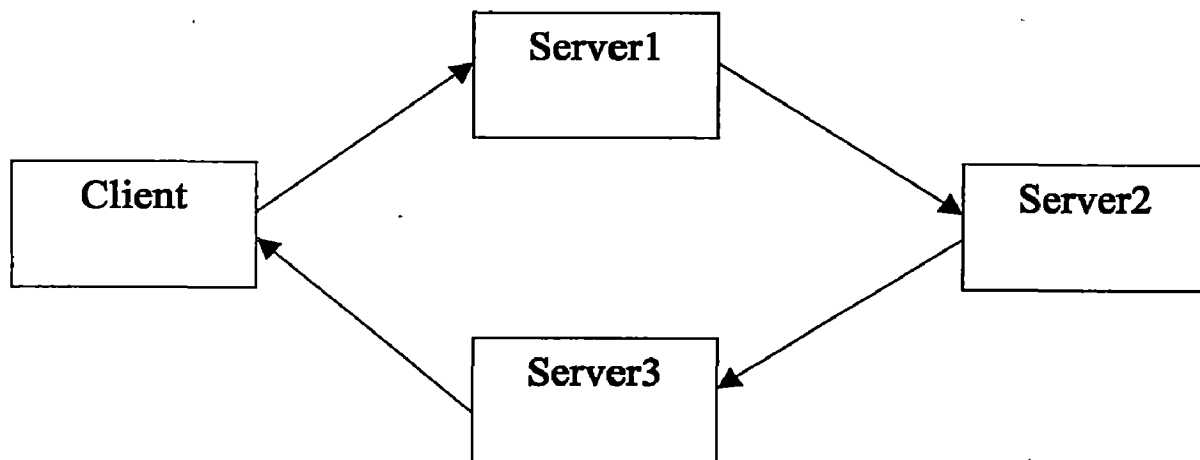


Fig.2.2.4.2: Communication using mobile agents

Mobile agents differ from traditional client/server application by moving themselves where the data are, instead of moving the data to where the application resides. This improves performance of the data collection by requiring less bandwidth.

Another advantage is the fact that mobile agents can perform their task asynchronously, or offline: the host of origin can initiate agents, tell them to migrate, go offline, and when back online, waiting for agents to come back.

2.2.5 Advantages of Mobile Agents

The main advantages of using mobile agents are

a) They reduce the network load.

Distributed systems often rely on communication protocols that involve multiple interactions to accomplish a given task. This is especially true when security measures are enabled. The result is a lot of network traffic. Mobile agents allow us to package a conversation and dispatch it to a destination host where the interactions can take place locally. Mobile agents are also useful when it comes to reducing the flow of raw data in the network. When very large volumes of data are stored at remote hosts, these data should be processed in the locality of the data, rather than transferred over the network. The motto is simple: move the computations to the data rather than the data to the computations.

b) They overcome network latency.

Critical real-time systems such as robots in manufacturing processes need to respond to changes in their environments in real time. Controlling such systems through a factory network of a substantial size involves significant latencies. For critical real-time systems, such latencies are not acceptable. Mobile agents offer a solution, since they can be dispatched from a central controller to act locally and directly execute the controller's directions.

c) They encapsulate protocols.

When data are exchanged in a distributed system, each host owns the code that implements the protocols needed to properly code outgoing data and interpret incoming data, respectively. However, as protocols evolve to accommodate new efficiency or security requirements, it is a cumbersome if not impossible task to upgrade protocol code properly. The result is often that protocols become a legacy problem. Mobile agents, on the other hand, are able to move to remote hosts in order to establish "channels" based on proprietary protocols.

d) They execute asynchronously and autonomously.

Often mobile devices have to rely on expensive or fragile network connections. That is, tasks that require a continuously open connection between a mobile device and a fixed network will most likely not be economically or technically feasible. Tasks can be embedded into mobile agents, which can then be dispatched into the network. After being dispatched, the mobile agents become independent of the creating process and can operate asynchronously and autonomously. The mobile device can reconnect at some later time to collect the agent.

e) They adapt dynamically.

Mobile agents have the ability to sense their execution environment and react autonomously to changes. Multiple mobile agents possess the unique ability to distribute themselves among the hosts in the network in such a way as to maintain the optimal configuration for solving a particular problem.

f) They are naturally heterogeneous.

Network computing is fundamentally heterogeneous, often from both hardware and software perspectives. As mobile agents are generally computer- and transport-layer-

independent, and dependent only on their execution environment, they provide optimal conditions for seamless system integration.

g) They are robust and fault-tolerant.

The ability of mobile agents to react dynamically to unfavorable 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 will be warned and given time to dispatch and continue their operation on another host in the network.

2.3 Mobile Agent Framework

As with any other communications-related activity, the general acceptance of mobile agents for network management activity will depend heavily upon standards. The Open Management Group (OMG) has already begun work in the area of mobile agents. The proposed standard attempts to be platform neutral and has each chunk of mobile code identify itself with a language, or execution environment requirement. The proposal identifies the need for mobile code regions, with gateways between them that provide an agent application virtual layer on top of the actual network. Fig.2.3 shows the mobile agent facility architecture.

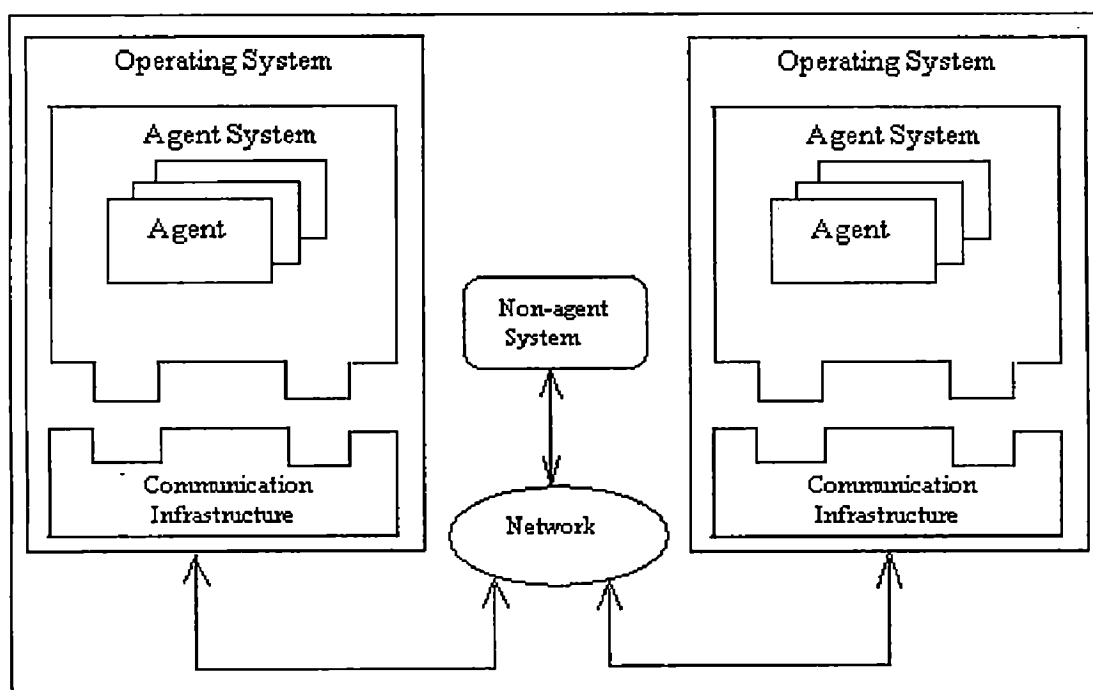


Fig.2.3: Mobile Agent Facility Architecture [4]

An agent region is defined as a set of agent systems that can access each other, possessing similar authority and identifying a default migration pattern. Mobile agent facilities include the storage and retrieval of agents, remote agent creation transfer and agent method invocation. The agent system is loaded on the operating system. There can be different agent system on one machine. Each agent system consists of the place and communication infrastructure, which is required for communication between two agent environments. Agent to agent communication is possible between two agents of same or different machine. The agent communication is based on the protocol used by the system or it is through its own protocol called as agent transfer protocol.

2.4 Java and Mobile Agents

When looking at mobile agents as a solution to some enterprise problem, a vital issue that must be satisfied is that of security and acceptability in a heterogeneous environment. If agents are to be allowed to move anywhere within some network architecture, whether the route followed is pre-known by each agent or dynamically determined, the system that propagates the agents must support both methods. Java is by far the most readily tailored system currently in widespread use that already supports the needs of mobile agent systems, with few drawbacks.

Java's network-centricity, sandbox security model, and platform independence make the language a perfect environment in which to development agent-based tools. One such tool, IBM's Aglet Workbench, provides a laboratory for creating Java-based mobile agent applications. The Aglet Workbench defines its own Java agent API (which has been submitted to standards committees) and provides a set of tools and samples for getting started. Java is inherently a network based API that has its roots in the one of the largest networks in the world. Since the agent technology is created from the Java API, it is readily available for immediate implementation in such environments. As Java has been essentially created for use on large heterogeneous networks, it also allows for platform independence, something that may be another issue for people looking to use the technology. Also, since Java was intended for use on the Internet, security has always been a major issue during its development. Also, Java already provides support for serialization, object transfer, and agent communication models.

However, there does exist some drawbacks to using the Java API as the root for a mobile agent system. Since Java is a programming API as any other, it cannot protect against such things as resource consumption by individual agents. For example, any agent running in a place can start looping and consuming part of that places resources without ceasing, thereby creating a sort of 'denial of service' class of problem.

2.5 Existing Mobile Agent Systems

Several academic and industrial research groups are currently investigating and building mobile agent systems. Much of the work in this area remains academic but this is likely to change. As users discover the power of delegating laborious tasks to agent technologies, their popularity will skyrocket. Companies such as General Magic, IBM, Crystalz, and others are already investing millions of dollars into developing commercial applications of mobile agents.

2.5.1 Telescript

Telescript, developed by General Magic, includes an object-oriented, type-safe language for agent programming. Telescript servers (which are called places) offer services, usually by installing stationary agents to interact with visiting agents. Agents use the go primitive for absolute migration to places, specified using DNS-based hostnames. The runtime system captures execution state at the thread level, so the agent resumes operation immediately after the go statement. Relative migration is also possible using the meet primitive. Co-located agents can invoke each other's methods for communication. 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. General Magic has now shelved the Telescript project and embarked on a similar, Java-based system called Odyssey [5] that uses the same design framework. In common with most other Java-based systems however, it lacks thread-level execution state capture.

2.5.2 Agent Tcl

Agent Tcl, developed at Dartmouth College, allows Tcl scripts to migrate between servers that support agent execution, communication, status queries and non-volatile storage. A modified Tcl interpreter is used to execute the scripts, and it allows the capture of execution state at the thread level. When an agent migrates, its entire source code, data and execution state is transferred. Migration is absolute, and the destination is specified using a location-dependent name. It is also possible to clone an agent and dispatch it to the desired server. Agents have location-dependent identifiers based on DNS hostnames, which therefore change upon migration. Inter-agent communication is accomplished either by exchanging messages or setting up a stream connection. Agent Tcl uses the Safe Tcl execution environment to provide restricted resource access. It ensures that agents cannot execute dangerous operations without the appropriate security mediation.

2.5.3 Voyager

This is a Java-based agent system developed by ObjectSpace [6]. A novel feature of Voyager is a utility called vcc, which takes any Java class and creates a remotely accessible equivalent, called a virtual class. An instance of a virtual class can be created on a remote host, resulting in a virtual reference that provides location-independent access to the instance. This mechanism is used for implementing agents. An agent is assigned a globally unique identifier, and an optional symbolic name during object construction. A name service is available, which can locate the agent, given its identifier or name. The virtual class provides a `moveTo` primitive, which allows the agent to migrate to the desired location. The destination is specified either using the server's DNS hostname and port number. Agent communication is possible via method invocation on virtual references. Agents can make synchronous, one-way, or future-reply type invocations.

2.5.4 Concordia

Concordia developed by Mitsubishi Electric [7], supports mobile agents written in Java. Like most Java-based systems, it provides agent mobility using Java's serialization and class loading mechanisms, and does not capture execution state at the thread level. Each agent object is associated with a separate Itinerary object, which specifies the

agent's migration path (using DNS hostnames) and the methods to be executed at each host. Concordia has extensive support for agent communication. Agent state is protected during transit, as well as in persistent stores, using encryption protocols. Each agent is associated with a particular user, and carries a one-way hash of that user's password.

2.6 Aglets

Aglets are a mobile agent technology developed at IBM's research laboratories in Japan. Aglets were developed as one implementation of a mobile agent system based on the Java API. All versions of JDK release of the Java API is supported for aglets and the ASDK (Aglet Software Development Kit), but with growth of the Internet marketplace advancements and updates are sure to be forthcoming. According to the IBM [8] white paper on Aglets, aglets are described as such:

Aglets are Java objects that can move from one host on the Internet to another. That is, an aglet that executes on one host can suddenly halt execution, dispatch to a remote host, and resume execution there. When the aglet moves, it brings along its program code as well as its state (data). A build-in security mechanism makes it safe to host untrusted aglets.

Aglets are a natural progression or evolution of Java technology on the Internet. First came the applet, allowing a graphical 'agent' to be downloaded into a client environment (browser) and execute there. Next came the servlet, another Java technology that allows a client program to upload program code to a server, which then instantiates and executes on the server for the benefit of any client wishing to see any results. Now Aglets have been created as the agents that can run on client or server, anywhere on the Internet or Intranets connected.

2.6.1 Where to use Aglets?

Some possible uses for mobile agents would be:

- **Electronic commerce:** Shoppers could make requests in a large marketplace via a mobile Aglet. The shopper's Aglet would travel to the marketplace and try to find the best match to their 'masters' request. In turn, sellers could invoke their own Aglets to pawn their wares to the shopper Aglets within that marketplace.

- **Secure brokering:** Since an Aglet is entirely self-contained, sensitive data such as credit-card numbers and online-brokerage identifications could be encrypted within an agent. These agents could travel to the consumer to get their latest buy/sell needs, and then return to the brokerage dealer to perform the transactions required.
- **Workflow applications:** An Aglet can follow an itinerary, or a planned route through the network in which it lives. This would allow a system where the Aglet could start with the engineer who plans a project and gives a product-list to it. Then the Aglet could travel to a Administrative assistant who could put in prices. Once this task is complete, the Aglet could then travel to the Manager who would approve the project as is or return the Aglet to the Engineer for revisal. Each participant in this scenario could see the Aglet with a different GUI, suited to match each viewing need.
- **Personal assistance.** The mobile agent's ability to execute on remote hosts makes it suitable as a "assistant" capable of performing tasks in the network on behalf of its creator. The remote assistant will operate independently of its limited network connectivity, and the creator can feel free to turn his or her computer off. To schedule a meeting with several other people, a user could send a mobile agent to interact with the representative agents of each of the people invited to the meeting. The agents could negotiate and establish a meeting time.
- **Distributed information retrieval.** Information retrieval is an often-used example of a mobile agent application. Instead of moving large amounts of data to the search engine so that it can create search indexes, you dispatch agents to remote information sources, where they locally create search indexes that can later be shipped back to the origin. Mobile agents are also able to perform extended searches that are not constrained by the hours during which the creator's computer is operational.
- **Parallel processing.** Given that mobile agents can create a cascade of clones in the network, one potential use of mobile agent technology is to administer parallel processing tasks. If a computation requires so much processor power as to that it

must be distributed among multiple processors, an infrastructure of mobile agent hosts could be a plausible way to get the processes out there.

- **Information dissemination.** Agents are able to disseminate information such as news and automatic software updates for vendors. The agents will bring the new software components as well as the installation procedures directly to the customer's personal computer and will autonomously update and manage the software on the computer.

AGLETS SPECIFICATION

Aglets are Java objects that can move from one host on the network to another. That is, an aglet that executes on one host can suddenly halt execution, dispatch to a remote host, and start executing again. When the aglet moves, it takes along its program code as well as the states of all the objects it is carrying. A built-in security mechanism makes it safe to host untrusted aglets.

3.1 ARCHITECTURE OVERVIEW

The Aglets architecture consists of two layers, and two APIs that define interfaces for accessing their functions[9].

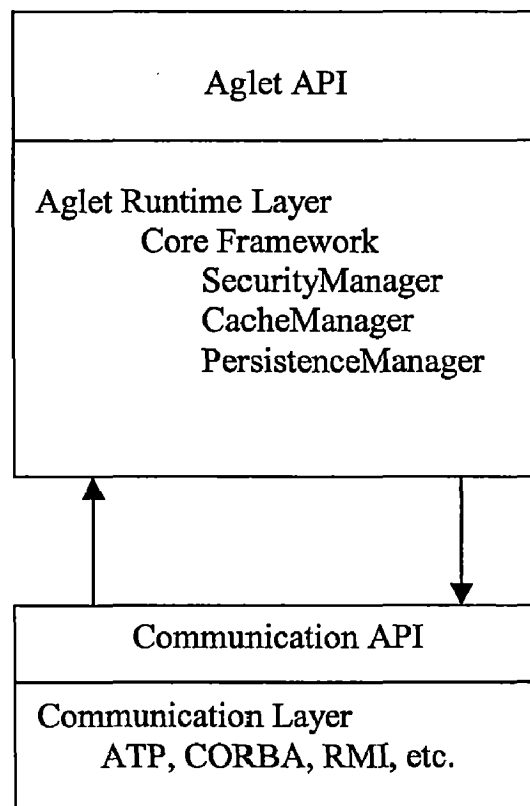


Fig.3.1: Aglet Architecture

The Aglets runtime layer is the implementation of the Aglet API, and defines the behavior of the API components, such as AgletProxy and AgletContext. It provides the fundamental functions for aglets to be created, managed, and dispatched to remote hosts. The communication layer is primarily responsible for transferring a serialized agent to a destination and receiving it. It also supports agent-to-agent communication and facilities for agent management.

3.1.1 Aglets Runtime Layer

The Aglets runtime layer implements Aglets interfaces such as AgletContext. It also consists of a core framework and subcomponents. The core framework provides the following mechanisms fundamental to aglet execution:

- Serialization and deserialization of aglets
- Class loading and transfer
- Reference management and garbage collection

The subcomponents are designed to be extensible and customizable because these services may vary depending on requirements or environments. For example, the PersistenceManager for applets may store deactivated aglets only in the memory, or else on the Web server if it can do so. In other cases, it may have to use the default security manager set by the Web browser.

- **PersistenceManager**

The PersistenceManager is responsible for storing the serialized agent, consisting of the aglet's code and state into a persistent medium such as a hard disk.

- **CacheManager**

The CacheManager is responsible for maintaining the bytecode used by the aglet. Because the bytecode of an incoming aglet needs to be transferred when the aglet moves to the next destination, the CacheManager caches all bytecode even after the corresponding class has been defined.

- **SecurityManager**

The SecurityManager is responsible for protecting hosts and aglets from malicious entities. It hooks every security-sensitive operation and checks whether the caller is permitted to perform it. There is only one instance of SecurityManager in the system, and it cannot be altered once it has been installed.

These components are defined as an interface or an abstract class, so server developers can implement these components for their own use and plug them into the runtime.

3.1.2 Communication Layer

The Aglets runtime layer itself has no communication mechanism for transferring the serialized data of an aglet to destinations. Instead, the Aglets runtime layer uses the communication API that abstracts the communication between agent systems. This API defines methods for creating and transferring agents, tracking agents, and managing agents in an agent-system- and protocol-independent way.

The current Aglets uses the Agent Transfer Protocol (ATP) as the default implementation of the communication layer. ATP is modeled on the HTTP protocol, and is an application-level protocol for transmission of mobile agents.

Agent Transfer Protocol

ATP is a simple application-level protocol designed to transmit an agent in an agent-system-independent manner. ATP offers a simple and platform independent protocol for transferring agents between networked computers. While mobile agents may be programmed in many different languages and for a variety of vendor specific agent platforms (consisting of virtual machines and libraries), ATP offers the opportunity to handle agent mobility in a general and uniform way:

- A machine hosting agents has an ATP-based agent service which is a component capable of receiving and sending agents from remote hosts via the ATP protocol. The agent service is identified by a unique address, independent of the specific agent platforms supported by the machine. A machine can run multiple agent services.
- A machine can host different types of agents, provided it supports the corresponding agent platforms.
- Any agent platform should include a handler of ATP messages.
- An ATP message carries sufficient information to identify the specific agent platform (at the receiver host) and calling its ATP handler to handle the message.

An ATP request consists of a request line, header fields, and a content. The request line specifies the method of the request, while the header fields contain the parameters of the request. ATP defines the following four standard request methods:

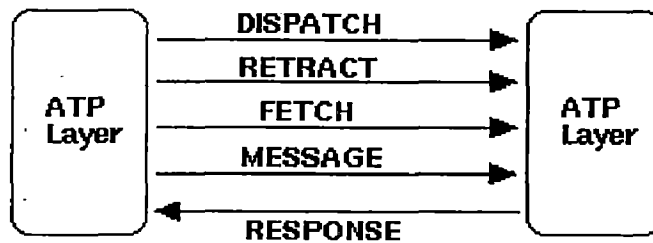


Fig: 3.1.2: Communication between agents

- **Dispatch**

The dispatch method requests a destination agent system to reconstruct an agent from the content of a request and to start executing the agent. If the request is successful, the sender must terminate the agent and release any resources consumed by it.

- **Retract**

The retract method requests a destination agent system to send a specified agent back to the sender. The receiver is responsible for reconstructing and resuming the agent. If the agent is successfully transferred, the receiver must terminate the agent and release any resources consumed by it.

- **Fetch**

The fetch method is similar to the GET method in HTTP; it requests a receiver to retrieve and send any identified information (normally class files).

- **Message**

The message method is used to pass a message to an agent identified by a agent-id and to return a reply value in the response. Although the protocol adopts a request/reply form, it does not lay down any rules for a scheme of communication between agents.

3.2 Aglet API Overview

The Aglet API defines the fundamental functionality of mobile agents. Fig3.2 shows the major interfaces and classes defined in the Aglet API and the relationship between these interfaces.

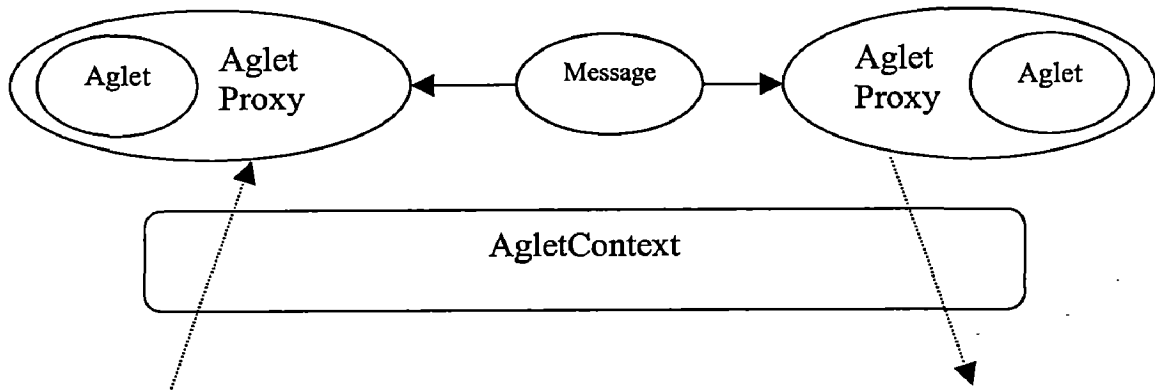


Fig.3.2: Aglet API

3.2.1 com.ibm.aglet.Aglet

The Aglet abstract class defines the fundamental methods (for example, `dispatch(URL)`) used to control the mobility and life cycles of mobile agents. All mobile agents defined in Aglet have to extend this abstract class. The `Aglet.dispatch(URL)` primitive causes an aglet to move from the local machine to the destination specified as its argument. The `Aglet.deactivate(long time)` primitive allows an aglet to be stored in secondary storage, and the `Aglet.clone()` primitive spawns a new instance of the aglet that has the same state as the original aglet. Fig.3.2.1 shows some primary methods and their semantics.

Method	Behavior
<code>Dispose()</code>	Dispose off the aglet
<code>Dispatch(url)</code>	Dispatch the aglet to the destination specified by the url
<code>Deactivate(long period)</code>	Instruct the aglet to store itself int a persistent medium
<code>GetAgletInfo()</code>	Get information on the aglet

Fig 3.2.1: Primary Methods of Aglet

3.2.2 com.ibm.aglet.AgletProxy

The AgletProxy interface object acts as a handle of an aglet and provides a common way of accessing the aglet behind it. Since an aglet class has several public methods that should not be accessed directly from other aglets for security reasons, any aglet that wants to communicate with other aglets has to first obtain the proxy object, and then interact through this interface. In other words, the aglet proxy acts as a shield object that protects an agent from malicious agents. When invoked, the proxy object consults the SecurityManager to determine whether the caller is permitted to perform the method. Another important role of the AgletProxy interface is to provide the aglet with location transparency. If the actual aglet resides at a remote host, it forwards the requests to the remote host and returns the result to the local host.

The AgletProxy can be obtained in the following ways:

- Get an enumeration of proxies in a context by calling the primitive `AgletContext.getAgletProxies()`.
- Get an AgletProxy for a given AgletID via either `AgletContext.getAgletProxy(AgletID)` or `Aglets.getAgletProxy(String contextName, AgletID)`.
- Get an AgletProxy object by message passing. An AgletProxy object can be put into the Message object as an argument, and sent to the aglet locally or remotely.
- Put an AgletProxy object into the context-property by `AgletContext.setProperty(String, Object)`, and share the proxy object.

The runtime library is responsible for providing the implementation of the AgletProxy interface; thus, aglet programmers do not have to implement this interface.

3.2.3 com.ibm.aglet.AgletContext

The AgletContext class provides an interface to the runtime environment that occupies the aglet. Any aglet can obtain a reference to its current AgletContext object via the `Aglet.getAgletContext()` primitive, and use it to obtain local information such as the address of the hosting context and the enumeration of AgletProxies, or to create a new aglet in the context. Once the aglet has been dispatched, the context object currently occupied is no longer available, and the destination context object is attached instead

when arrived. The runtime library is responsible for providing the implementation of this interface; thus, aglet programmers do not have to implement this interface.

3.2.4 com.ibm.aglet.Message

Aglet objects communicate by exchanging objects of the Message class. A message object has a String object to specify the kind of the message and arbitrary objects as arguments. A message can be sent to the aglet by calling `Object AgletProxy.sendMessage(Message msg)` and it is passed as an argument to `Aglet.handleMessage(Message msg)`.

3.3 Aglet Object and its Life Cycle

The `com.ibm.aglet.Aglet` class provides the basic functionality for a mobile object, and every aglet (aglet objects) has to be an instance of a subclass of it.

To use an aglet, program first has to instantiate it. There are two ways to create a new instance of an aglet. The first is to instantiate a completely new aglet from class definitions by calling `AgletContext.createAglet(URL codebase, String name, Object init)`. This primitive creates a new instance within the specified context and initializes it if necessary, then invokes `Aglet.onCreate(Object init)` on the created object along with the initializer object passed to the `createAglet` primitive. The other way is to create a copy of an existing aglet by using the `Aglet.clone()` primitive. The cloned aglet has the same state as the original one but has a different `AgletID` object, and thus a distinct identity.

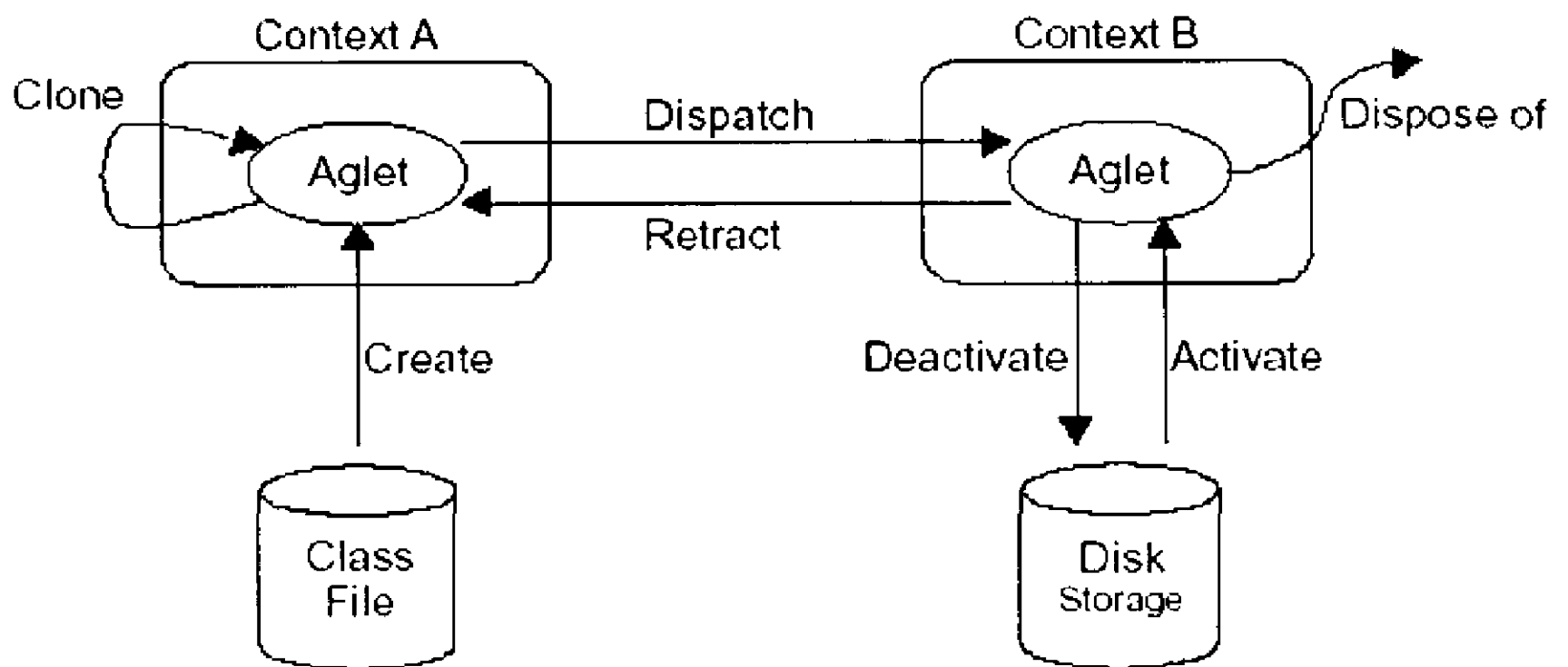
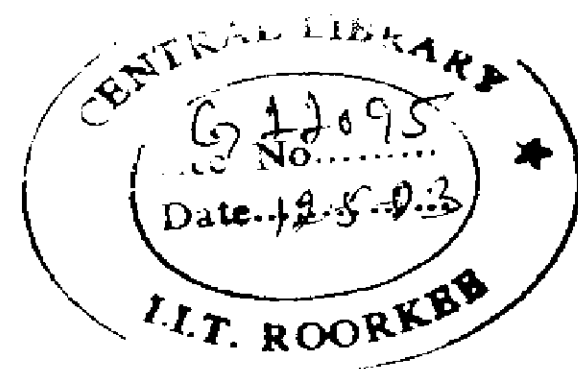


Fig.3.3 :Aglet Lifecycle[9]



Once created, an aglet object can be dispatched to and/or retracted from a remote server, deactivated and placed in secondary storage, then activated later.

An aglet can dispatch itself to a remote server by calling the `Aglet.dispatch(URL dest)` primitive. To be more precise, an aglet occupies the aglet context and can move from this context to others during its execution. Because the server may serve multiple contexts within one Java VM, and one host may serve multiple servers in one host the context are named as the following set

- the address of the host, typically IP-address.
- the port number to which the server is listening.
- the name of context within the server.

Example:

```
atp://aglets.ibm.com:4434/context_name
```

Dispatching causes an aglet to suspend its execution, serialize its internal state and bytecode into the standard form and then to be transported to the destination. On the receiver side, the Java object is reconstructed according to the data received from the origin, and a new thread is assigned and executed.

Aglets can be persistent. Since a mobile aglet needs to be serializable into a bit-stream, all mobile aglet can be persistent in nature. The `Aglet.deactivate(long timeout)` primitive causes an aglet to be stored in secondary storage and to sleep for a specified number of milliseconds. After the given time has passed or another program has requested its activation, the aglet is activated within the same context where as that in which it was deactivated.

Unlike normal Java objects, which are automatically released by garbage collector, an aglet object, since it is active, can decide whether or not to die. If you call the `dispose()` method to kill the aglet, `onDisposing()` is called to perform the finalization suitable for the current state of the aglet .

3.4 Security in Aglets

Security is essential to any mobile agent system, because accepting a hostile agent may lead to your computer being damaged or your privacy intruded upon. For secure agent execution, the agent system must provide the following security services:

- Authentication of the Sender, the Manufacturer and the Owner of the Agent.
 - Who is responsible for this agent?
 - Who is responsible for the agent code?
 - Has the agent (code and state) been tampered with?
- Authorization of the Agent (or Its Owner)
 - What can this agent do? (e.g, can this agent access files?)
- Secure Communication between Agent Systems.
 - Can the agent protect its privacy?
- Non-repudiation and Auditing.
 - How can we ensure that a deal has been actually carried out?
 - Security-sensitive activities of agents must be recorded, and an administrator must be able to audit them.

In mobile agent systems, agents must present proper user identities so that agent systems can control them according to the access rights of the users and the agent's manufacturers. It is therefore important for agent systems to be able to authenticate an agent's user and manufacturer. The former is much more difficult than the latter. It is reasonably easy to identify the manufacturer by code-signing. However, it is difficult to verify the ownership of the agent since the state of the agent varies during its travels, and it is practically impossible to sign the state part of the agent.

Aglets uses an organizational approach whereby all agent systems in a certain domain are deemed trustworthy, and evaluates the authenticity of the agent depending on the domain in which it has been roaming around. A user first authenticates himself to the system, and the system then issues the credentials of the user's agent. The agent system then evaluates the authenticity of the credentials, to determine whether or not they were issued within the same domain. It may downgrade the authenticity or simply deny access, depending on conditions such as where the agent has traveled and so forth. Host authentication is used to identify the domain to which the communicating host belongs.

Although the current Aglets does not fully support these services because of the limited support for encryption in JDK, it does provide a reasonable level of security to make it safe to use mobile agent applications. The following security features are supported in the latest Aglets runtime:

- Authentication of users and domains.
- Integrity checked communication between servers within a domain.

SYSTEM DESIGN AND IMPLEMENTATION

4.1 IBM's Aglets Software Development Kit

The Aglets Software Development Kit (ASDK) is an implementation of the Aglet API. It includes Aglet API packages, documentation, sample agents, and the Tahiti aglet server. This Aglet Workbench works on JDK1.1 or higher versions. It is qualified to run on Win95/NT and SPARC/Solaris 2.5.

The Aglet API is an agent development kit. It is a set of java classes and interfaces that allows the user to create mobile Java agents. Once the user has written an aglet, it will run on every machine that supports the Aglet API. The user need not concern about the underlying hardware or operating system.

4.1.1 Tahiti Server

For launching the agents first of all the user need to start the aglet server. Aglet server is started using the script file 'c:\aglets-2.0.1\bin\agletsd'. This aglet server will invoke an aglet viewer, named Tahiti, for managing aglets. The Tahiti window is as shown in Fig.4.1.1. The working of the Tahiti server is given in detail in Appendix A.

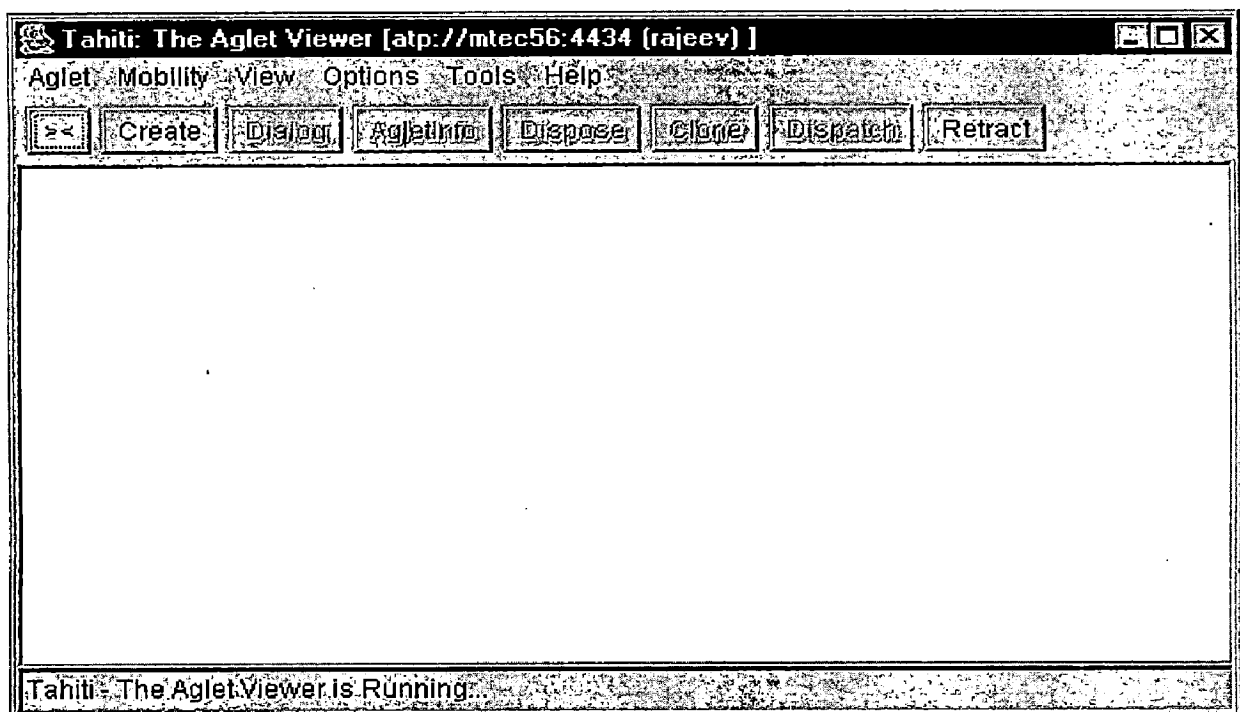


Fig.4.1.1: Tahiti Server

4.2 Design of Mobile Agents on Aglet platform

A simple agent consists of basically the main class and two methods `onCreation()` and `run()`. First start by importing the `aglet` package, which contains all the definitions of the Aglet API. Next define the `MyFirstAglet` class, which inherits from the `Aglet` class:

```
import com.ibm.aglet;
public class MyFirstAglet extends Aglet {
    //Put aglet's methods here....
}
```

For example, if you want your aglet to perform some specific initialization when it is created, you override its `onCreation` method:

```
public void onCreation(Object init) {
    //Do some initialization here....
}
```

When an aglet has been created or when it arrives in a new context, it is given its own thread of execution through a system invocation of its `run` method. The `run` method is called every time the aglet arrives at or is activated in a new context. So the `run` method becomes the main entry point for the aglet's thread of execution.

```
public void run() {
    //Do something else here...
}
```

`run()` method can be used to let the aglet dispatch itself to some remote context by calling its `dispatch` method with the Uniform Resource Locator (URL) of the remote host as the argument. This URL should specify the host and domain names of the destination context, and the protocol (`atp`) to be used for transferring the aglet over the network.

```
dispatch(new URL("atp://90.0.0.101:5000"));
```

In the dispatch method basically the aglet will disappear from the current host machine and reappear in the same state at the specified destination. First, a special technique called object serialization is used to preserve the state information of the aglet by making a sequential byte representation of the aglet. Next, this representation is passed to the underlying transfer layer that brings the aglet (byte code and state information) safely over the network. Finally, the transferred bytes are de-serialized to recreate the aglet's state.

4.3 Agents

The agent system for software distribution will consist of agents for directory and file listing, distribution, retrieving information of a system, and deletion.

4.3.1 System Property Agent

System property agent will bring the system information of the computer on which it is sent. Information will include operating system name, its version, user name, Java version, current working directory, etc. This information helps in deciding whether the software is compatible with the user's system or not. Fig.4.2.1 shows how a sender (owner of the software) sends a SysProAgent to the client side. The agent returns back carrying information regarding client's computer.

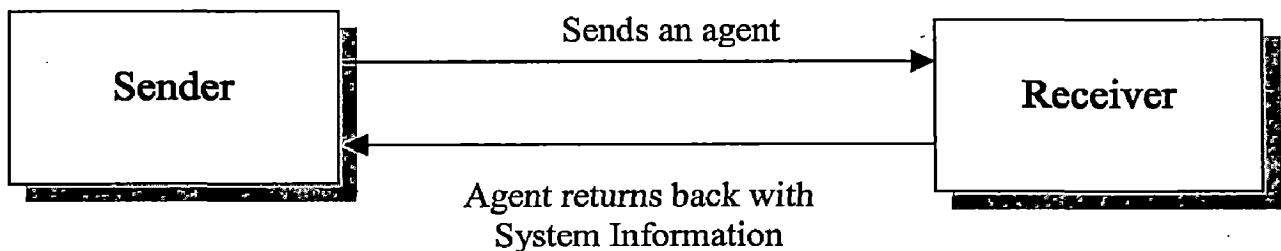


Fig.4.2.1: SysProAgent

4.3.2 Listing Agent

Listing agent will be able to recover the directories and files of the system on which our agent is sent. This information is helpful in installing the software in the desired directory. Fig.4.2.2 shows how this agent works.

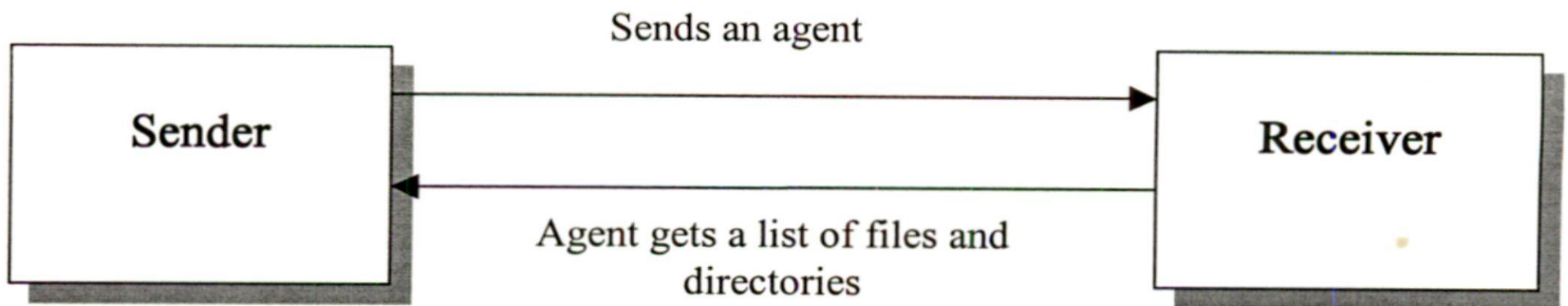


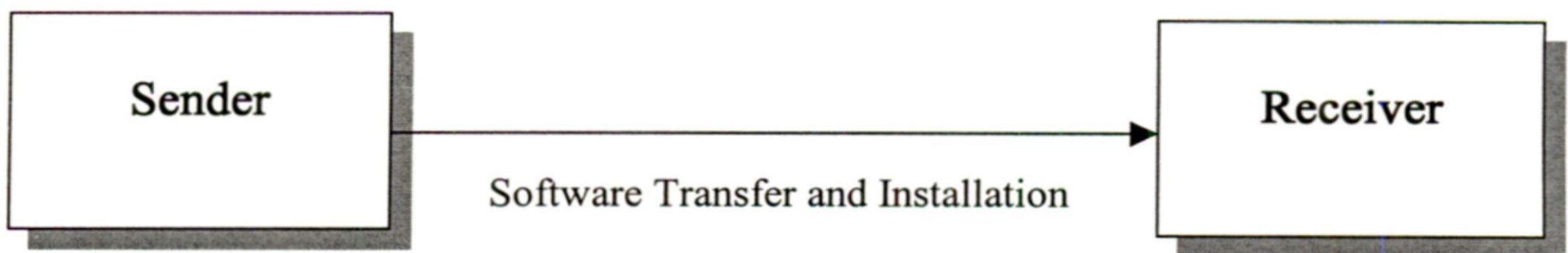
Fig.4.2.2: Listing Agent

During the onDispatching() or onCreate() method user can set the path of the directory he is looking for. At the onArrival() method the agent stores the directories and files of the directory specified. So when user retracts the agent, it will bring the list to the server.

4.3.3 Distribution Agent

Distribution agent will be the main agent, which will do the most important task of carrying the software to other hosts.

Fig.4.2.3: Distribution Agent



Before dispatching, store the file to be transferred to a buffer. This can be also implemented in the onDispatching() method. In the onArrival() method the agent then copies the data from that buffer and stores that in a folder on which the agent has been sent. Provision has also been made to execute the software once it is fully copied on the other host.

sent. Provision has also been made to execute the software once it is fully copied on the other host.

4.3.4 Delete Agent

Fig.4.2.4 shows the working of delete agent. This agent will go to the destination and delete the file specified. This is also a part of software distribution. Sometimes need may arrive to delete the original file after it has been installed, so this agent will come handy.

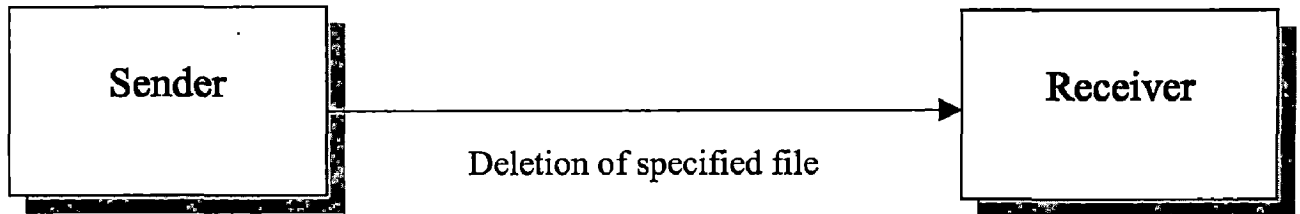


Fig.4.2.4: Delete Agent

RESULTS AND DISCUSSION

5.1 Tahiti Interface and Creation of Aglet

5.1.1 Creation of Aglet

Push the "Create" button or select the "Create" item in the "Aglet" menu on Tahiti. The panel as shown in fig.5.1.1 will appear.

Fill in the class name

Fill in the "Aglet name" field with the class name of the aglet, such as "sd.Diglet".

Fill in the URL

Case 1: Creating a local aglet

If you want to create an aglet from your local class, you do not have to specify a source URL. Please leave the "Source URL" field blank.

Case 2: Creating a remote aglet

Otherwise, you should specify the codebase as a URL for the remote class in the "Source URL" field.

After setting the aglet name, push the "Create" button on the panel, and you will find that a new list item appears in Tahiti.

Registering an aglet in the Aglet List

The "Add to List" button is used to register the class name in the "Aglet List" in this panel.

The "Delete" button is used to remove an aglet item from the list.

You can recall the aglet's name and its codebase by simply clicking the list item.

Creating an aglet

The "Create" button is used to create an aglet.

The "Reload Class and Create" button is used to create an aglet after reloading its aglet class. (Even though the class definition is not modified, another classloader will be created.)

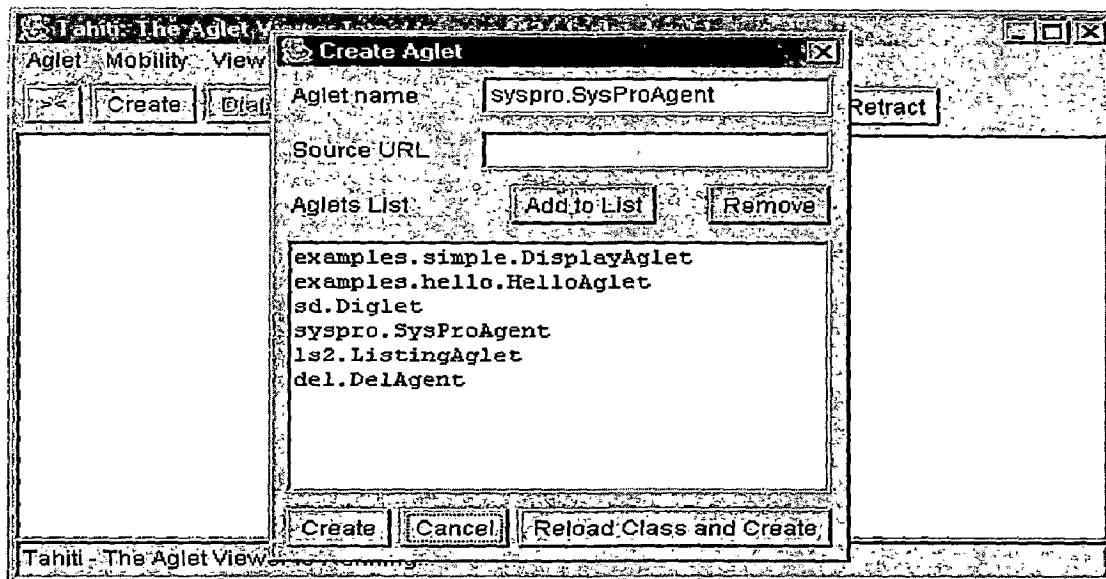


Fig.5.1.1: Creating aglets in Tahiti

Fig.5.1.1 shows how an aglet is created. After compiling the aglet program, the name of the aglet and its source URL is given in the Create Aglet frame. This way an agent is created.

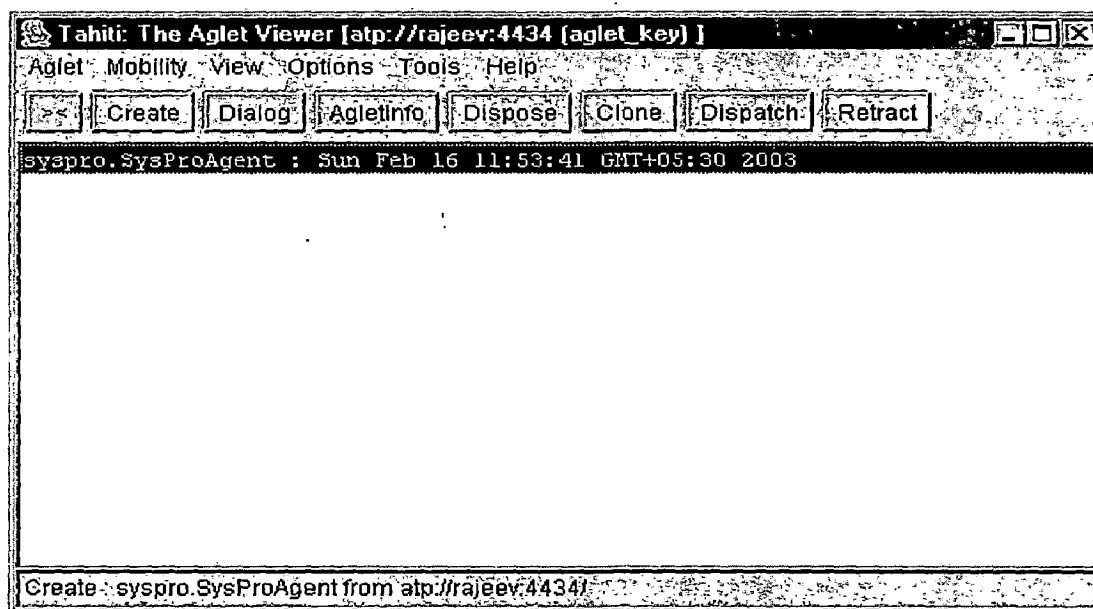


Fig.5.1.2: Created Aglet

Fig.5.1.2 shows an active agent. All the agents are displayed in the list box in Tahiti. By selecting one of the list items, you can control the corresponding agent (by dispatching it, deleting it, requesting a dialog with it, etc.).

5.2 Software Distribution Agents

The software distribution process consists of four agents as described in chapter 4. The interfaces and working of these agents is described below.

5.2.1 System Property Agent

System property agent is created after adding it to Tahiti's agent list. When this agent is created a GUI as shown above appears with all the fields blank (fig.5.2.1.1). The address IP address of the system on which the software has to be installed is entered in the "Address" field and the send button is pushed. The agent is dispatched to the desired destination. The agent after reaching the destination collects the required system information. The agent at destination system is as shown in fig.5.2.1.2. This agent is then retracted back to the sender as shown in fig.5.2.1.3. After reaching back to the sender the GUI as shown in fig.5.2.1.4 appears showing the information collected from the destination node.

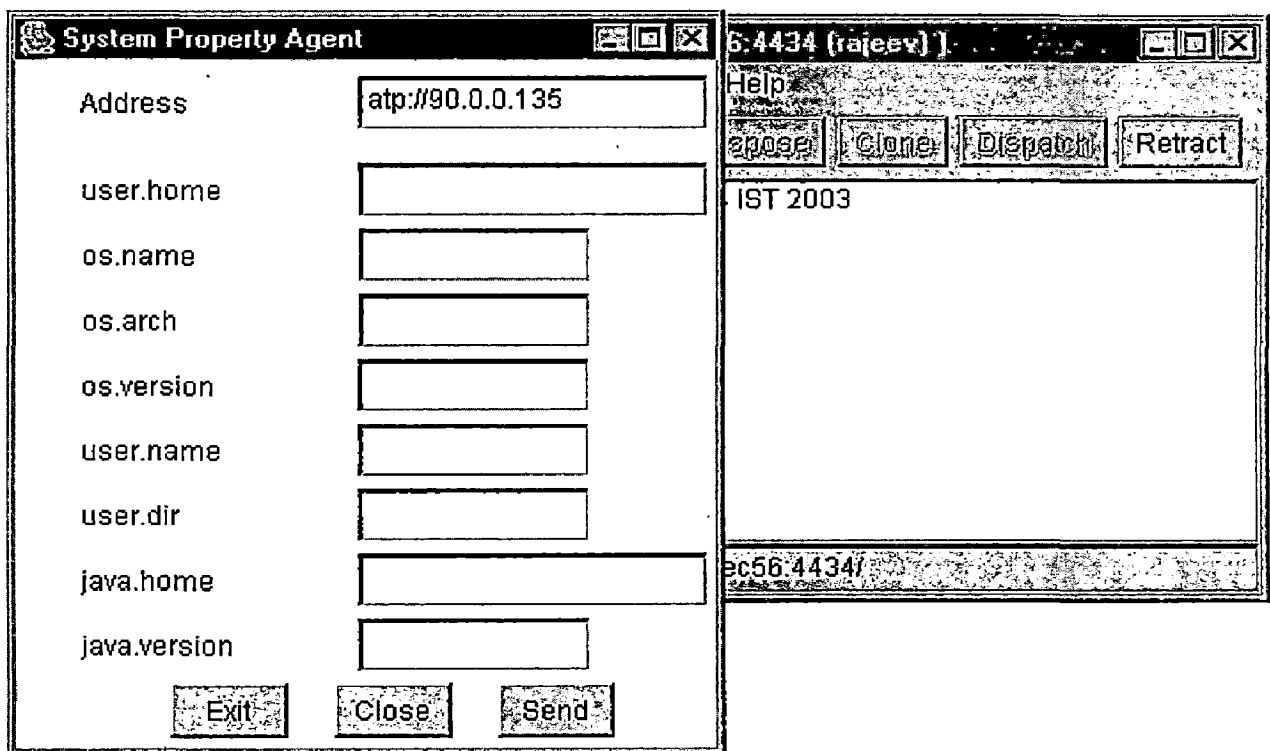


Fig.5.2.1.1: System Property Agent after initialization

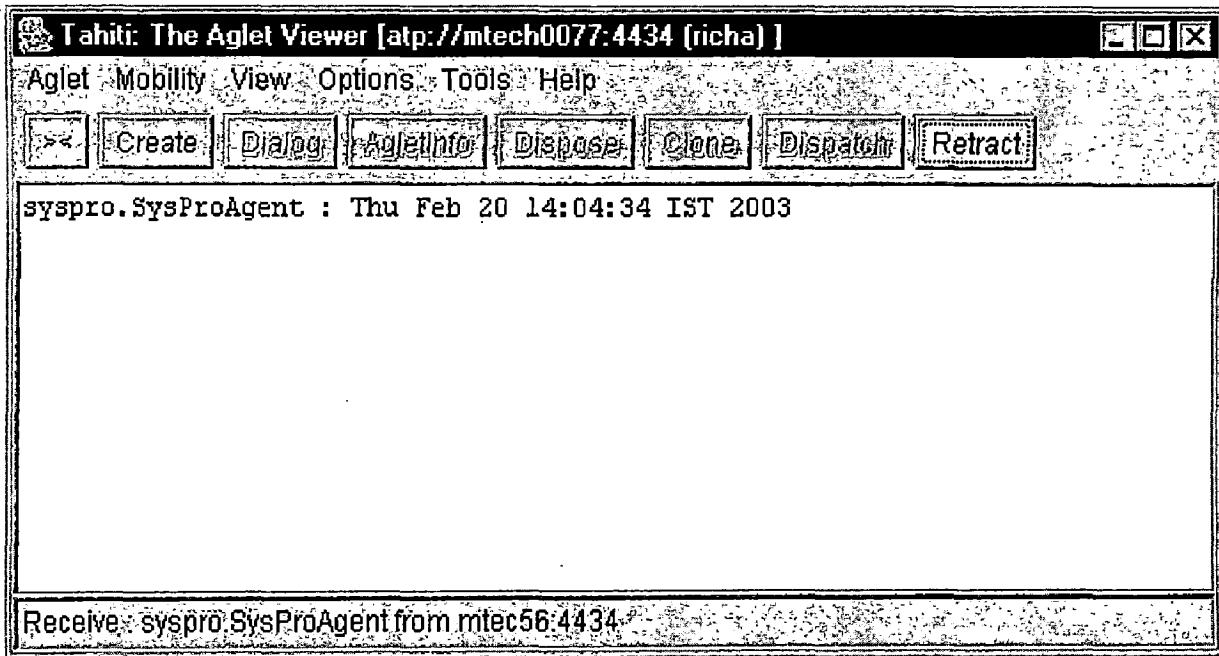


Fig.5.2.1.2: System Property Agent at destination node

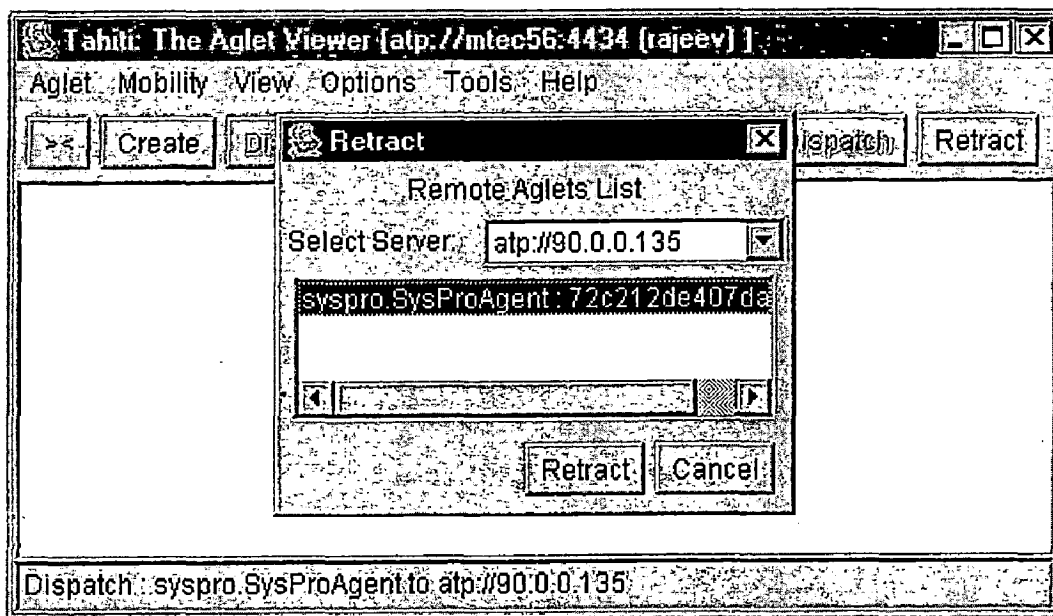


Fig.5.2.1.3: Retracting System Property Agent

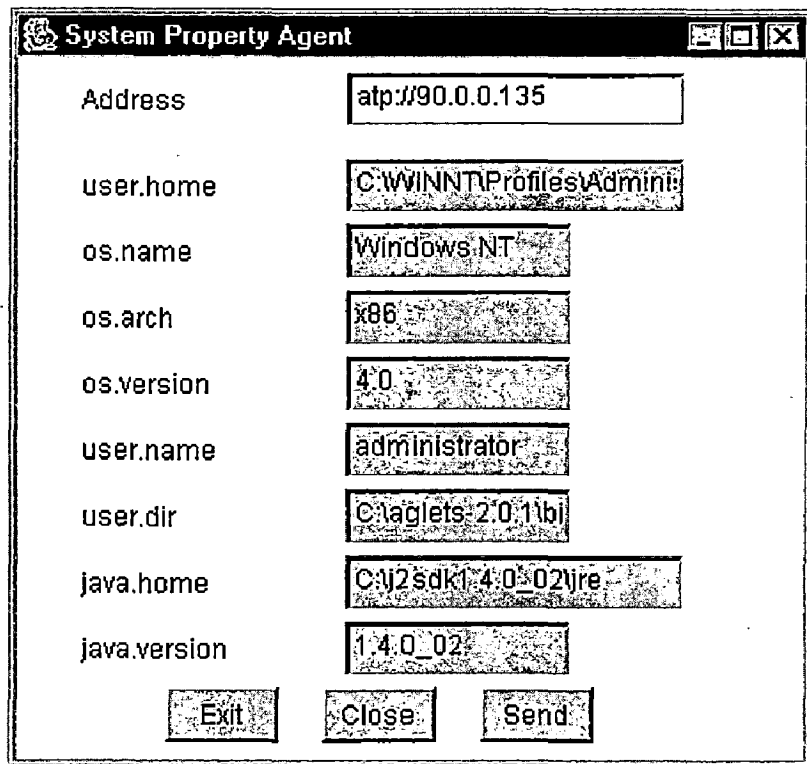


Fig.5.2.1.4: System Property Agent with results

5.2.2 Listing Agent

Listing agent is used to know the contents (files and directories) of the destination node. After initialization of the agent, the address of the destination node is entered in the "Address" field and the name of the required drive or directory is entered in the "Path" field as shown in fig.5.2.2.1. Once this is done, then the agent is dispatched to the destination node (fig.5.2.2.2). When the agent is retracted, it brings the contents of the directory mentioned in the "Path" field. This is shown in fig.5.2.2.3.

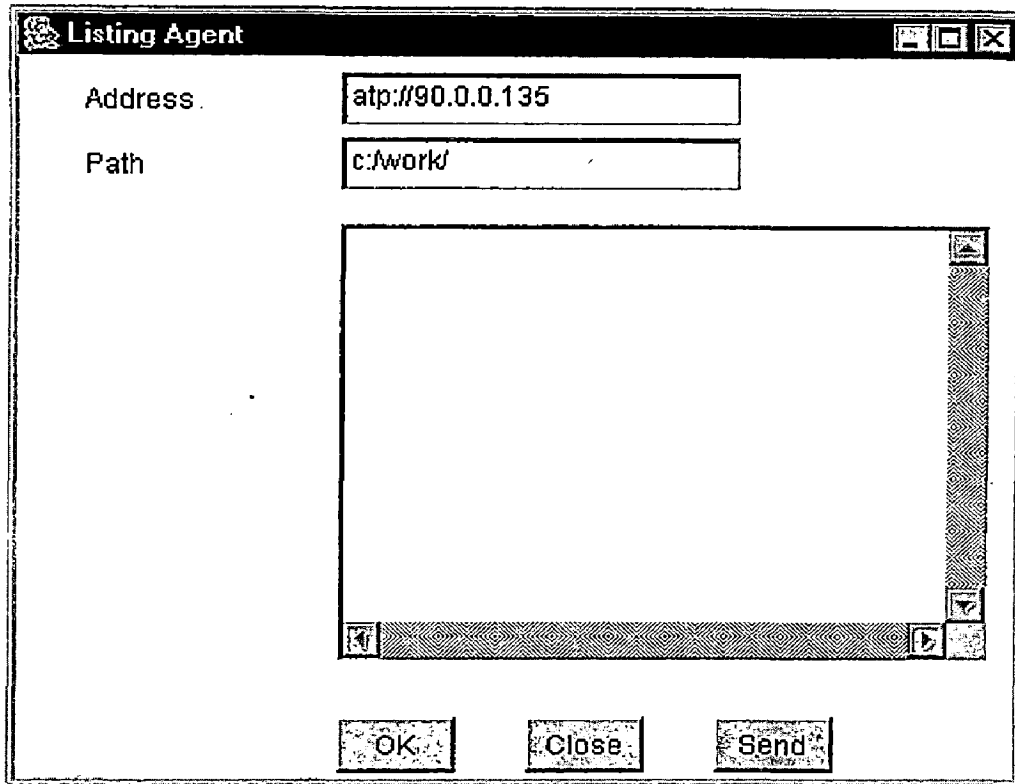


Fig.5.2.2.1: Listing Agent

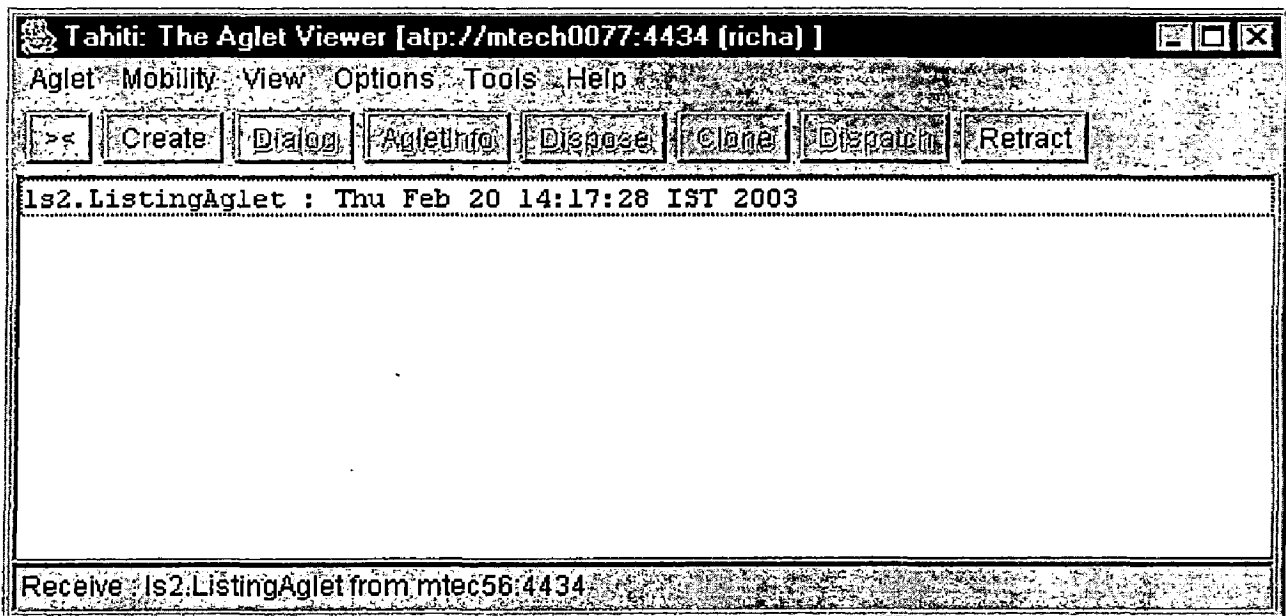


Fig.5.2.2.2: Listing Agent at destination node

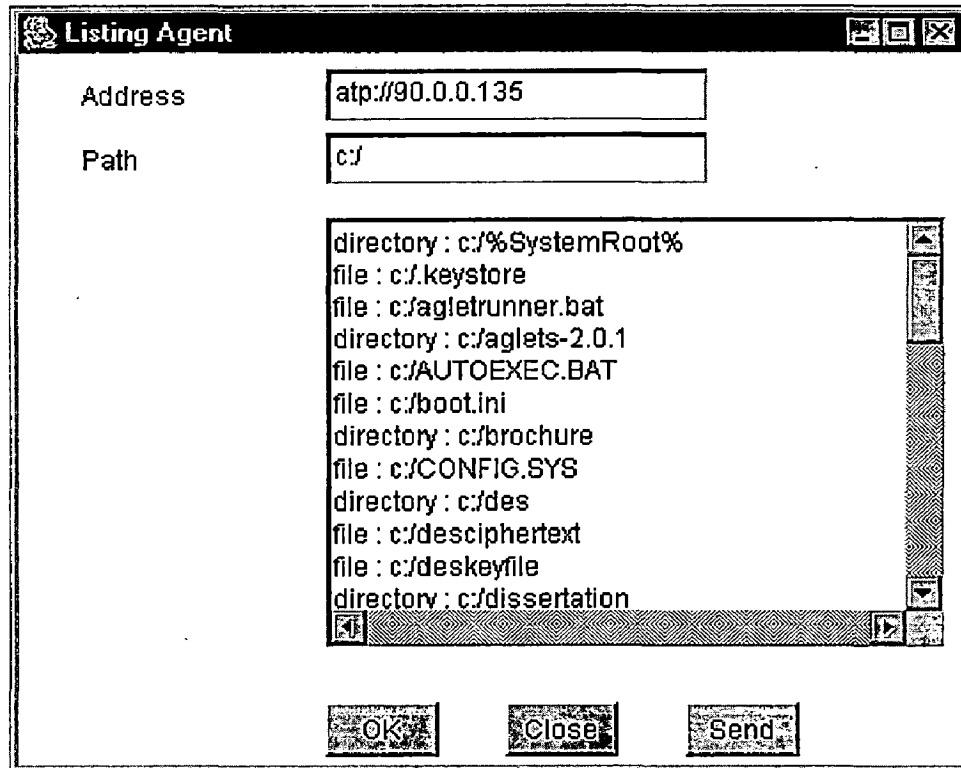


Fig.5.2.2.3: Listing Agent with results

5.2.3 Distribution Agent

Fig.5.2.3.1 shows the distribution agent. The path of executable file, which is to be transferred, is written in the "File Name" field. The file, which is to be transferred, can be selected using the browse option (fig.5.2.3.2) or directly the path of the file can be written in the respective textfield. Several clones of the agent can be created, but once user attaches a file to the agent, another file to the clone cannot be attached at the same time. First user will have to dispatch the agent with the attached file and then attach the file to the clone.

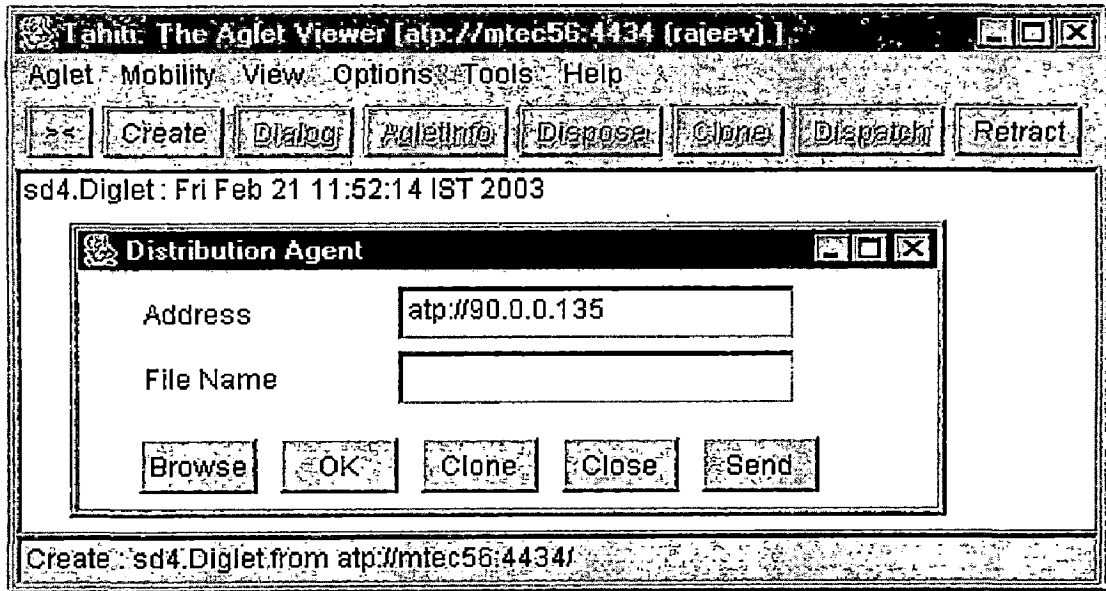


Fig.5.2.3.1 Distribution Agent

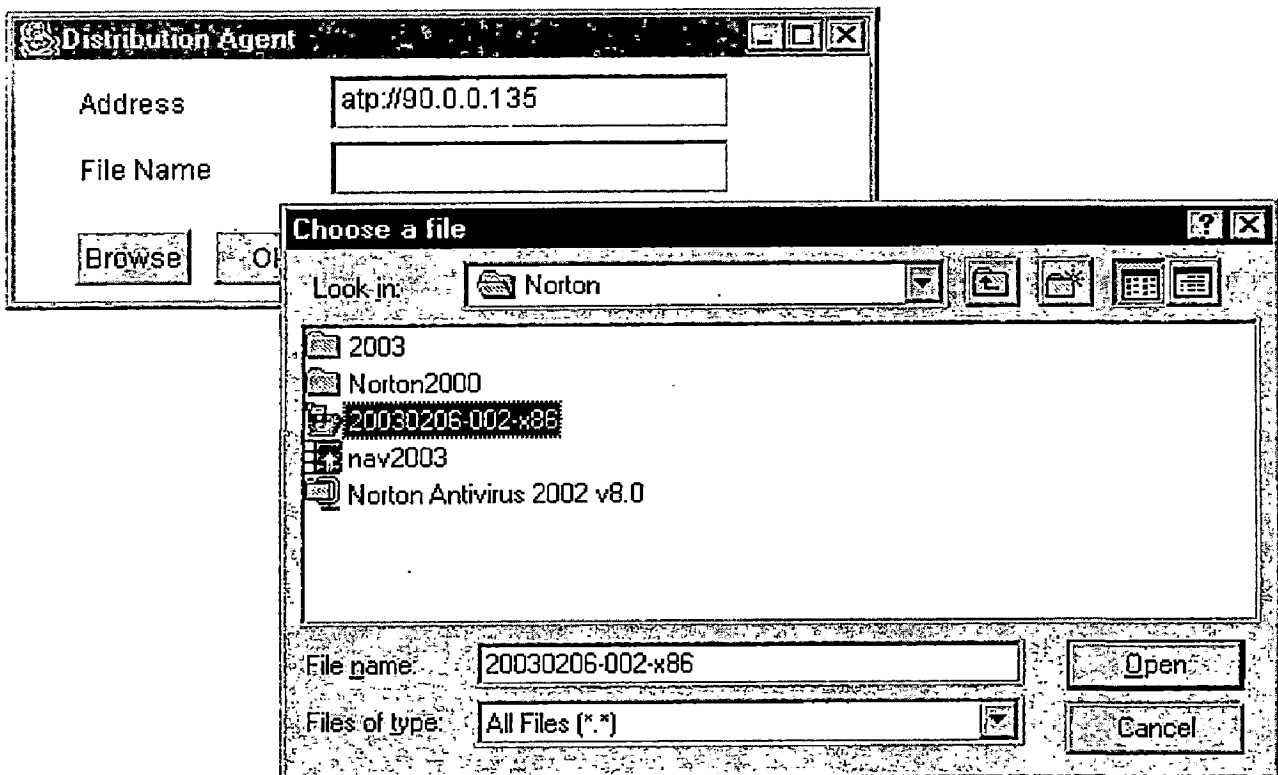


Fig.5.2.3.2: Selecting a file using Distribution Agent

5.2.4 Delete Agent

Delete agent performs a simple task of deleting a specified file on the system to which this agent is sent. The path of the file is mentioned in the "File Name" field as shown in Fig.5.2.4.

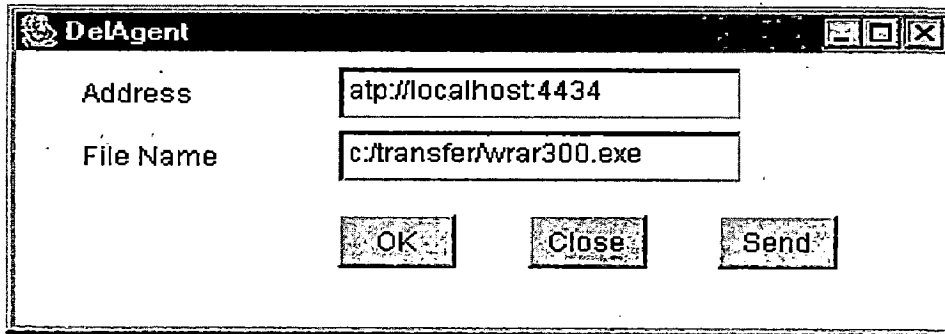


Fig.5.2.4: Delete Agent

CONCLUSION

A software distribution system has been created using the mobile agent technology. It has been created using Aglets, a programming language for mobile agents. The system provides many useful features such as retrieving system information, files and directory listing, and deletion. But the main task is of course software distribution.

The system consists of four agents

- Software Distribution Agent

As the name suggests this agent can carry files to a specified location. Feature like cloning is also provided in the interface.

- System Property Agent

This agent brings the some of the system properties of the specified system.

- Listing Agent

This agent is useful in getting the files and directory listing of the specified node. This is especially helpful if user wants to transfer the package in some specified directory.

- Delete Agent

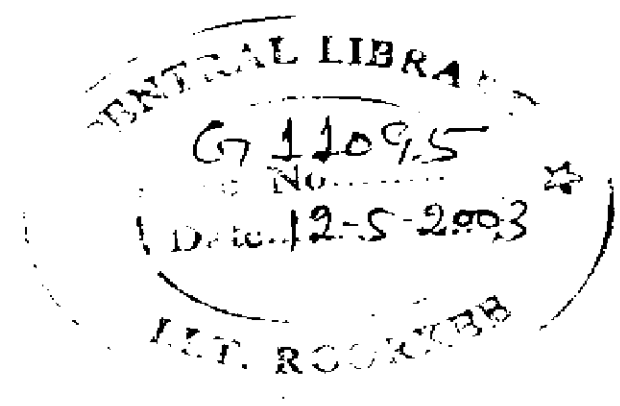
This agent is able to delete any specified file.

One of the major drawback for using the mobile agents is that the Aglet SDK platform has to be present on the system on which the agent has to run.

Presently the distribution agent can carry software and initiate its installation, so it is useful if the software package is executable from start to end. In further work, distribution agent can be created which can install any software completely and without the interaction of the user.

REFERENCES:

- 1 Danny B. Lange, Mobile Objects and Mobile Agents: The Future of Distributed Computing?, General Magic Inc., <http://www.acm.org/~danny>
- 2 David Kotz and Robert S. Gray, Mobile Agents and the Future of the Internet, Dartmouth College, May 15, 1999.
- 3 Roberto Silveira Silva Filho, Mobile Agents and Software Deployment, University of California.
- 4 A. Bieszczad, B.Pagurek, and T.White, Mobile Agents for Network Management, IEEE Communications Surveys, September 1998.
- 5 General Magic, Inc. Odyssey web page
<http://www.genmagic.com/technology/odyssey.html>, 1997.
- 6 ObjectSpace Voyager Core Package Technical Overview. Technical report, ObjectSpace, Inc., July 1997.
- 7 Concordia: An Infrastructure for Collaborating Mobile Agents. In Proceedings of the 1st International Workshop on Mobile Agents, April 1997.
- 8 D.B. Lange and M.Oshima, Aglets, paper available at
<http://www.trl.ibm.co.jp/aglets/whitepaper.htm>, 1997.
- 9 M. Oshima, G. Karjoth, K. Ono, Aglet Specification 1.1 Draft, available at
<http://www.trl.ibm.com/aglets/spec11.htm>, 8 September 1998.
- 10 D.B. Lange and M.Oshima, Programming and Deploying Java Mobile Agents with Aglets™, Addison-Wesley, a working draft Programming Mobile Agents in Java, is available at <http://www.trl.ibm.co.jp/aglets>, 1997.



Tahiti Menu Structure

Tahiti provides the functions for handling agents and for controlling the server, which can access from the following menu items.

A.1 Aglet: for handling aglets

Create...	Create an Aglet.
Dialog...	Sends a request to an aglet to open its dialog panel.
Dispose...	Destroys the agent.
Clone...	Make a copy of the agent.
AgletInfo	Shows the properties of the agent.
Kill	Destroys the agent. Aglet does not call onDisposing().
Exit	Shutdown the server.

A.2 Mobility

Dispatch	Send the aglet to another server.
Retract	Retract a dispatched aglet from another server.
Deactivate	Deactivate the aglet with time.
Activate	Activate a deactivated aglet.

A.3 View

Memory Usage	Show the memory usage amount
Log	Show the logged records of agents behavior on this server.

A.4 Options

General Preference	Font, Startup Aglet, Cache.
Network Preference	Set parameters for Http Tunneling, Authentication, etc.
Security Preference	Set access privileges for aglets. Specify File System, Network Access, and others.
Server Preference	Server Setting.

A.5 Tools

Invoke GC	Start garbage collection.
Threads	Display the current thread information on the console.

A.6 Help

About Tahiti...	Information about this aglet viewer.
About Aglets...	Information about the aglet library.
Release Notes	Open release notes.
Aglets Home Page	Open Aglets page on the WWW
Feedback	Open Aglets feedback page on the WWW
Bug Report	Open Aglets bug report page on the WWW
Frequently Asked Questions	Open Aglets FAQ page on the WWW

A.7 Customization with dialog interface

You need to specify some parameters for your aglet server. Setting with the default values does not provide the full functionality of this server. By tuning up Tahiti with those parameters, your aglet server works effectively.

A.7.1 General Preferences

Font: Defines the size of the font in Tahiti window. The change of the setting becomes effective immediately.

List View: Defines the order of the Aglet items in the list box of the Tahiti window. The change of the setting will be reflected after clicking one of the items on the list box.

Startup: You can specify an aglet that automatically starts up when Tahiti is started. (The Launch Startup Aglet checkbox enables this function.)

Cache Control: Clearing up the class cache.

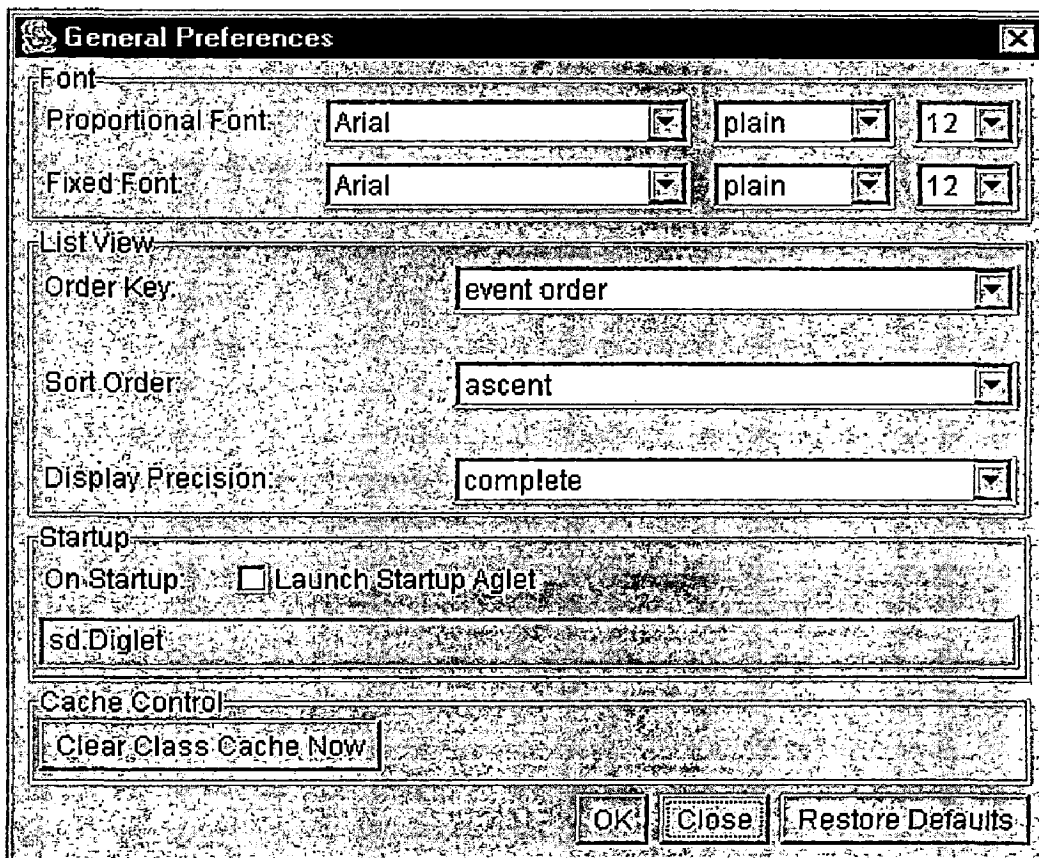


Fig.A.7.1: General Preferences

A.7.2 Network Preferences

Http Tunneling: In case you are protected by a firewall, you can specify a http proxy server to access the information at the outside of the firewall. We call this method as Http Tunneling. Check “Use HTTP Proxy” and Specify your http proxy information like a setting in a world wide web browser.

Authentication: This is a switch for security options. When you enable Authentication, Tahiti have to keep at least one secret file. Tahiti can communicate with each other only if the other aglet server has (can access) the same secret file.

Accept HTTP Request as a Message: We can create an aglet, which has a URL and returns html. When we enable this option, aglet receives HTTP request as a message.

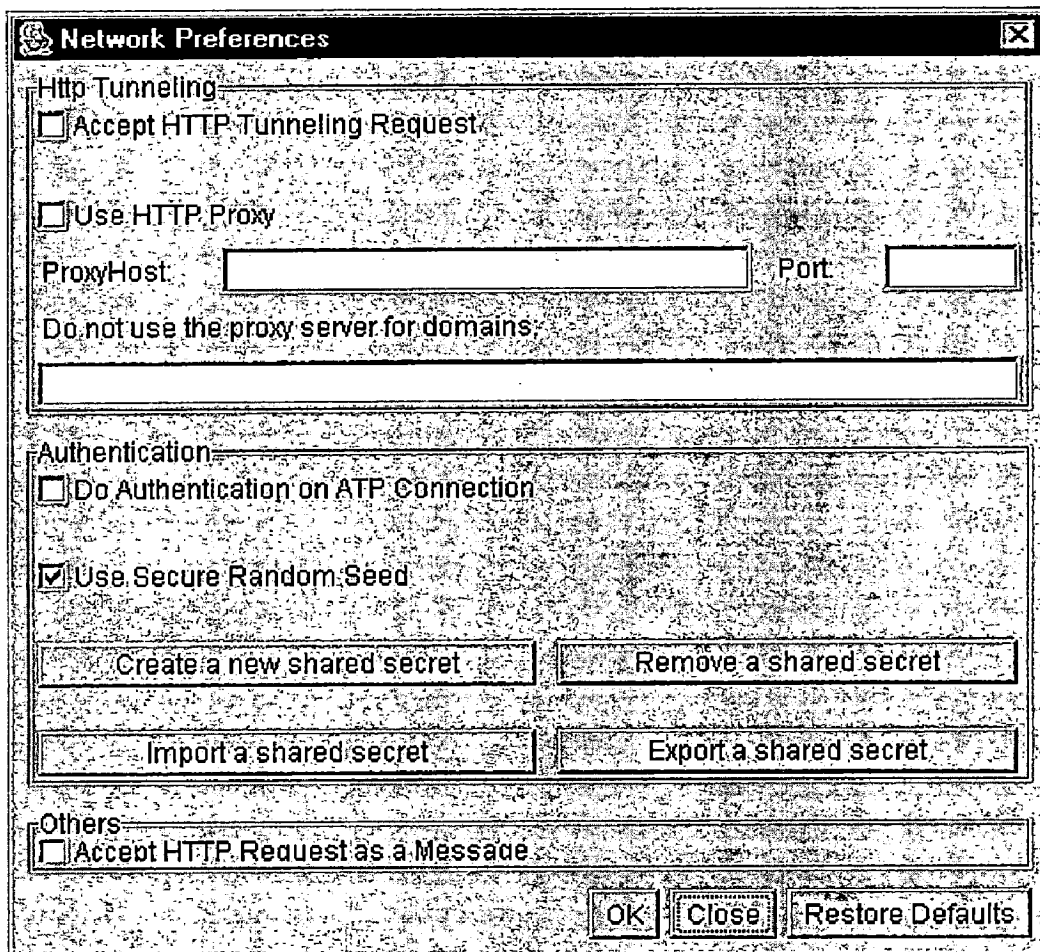


Fig.A.7.2: Network Preferences

A.7.3 Security Preferences

We have a dialog box for specifying security. User can specify the access privilege for the following items.

- FileSystem: Restricts the access to the files. "File/Directory" field is defined by absolute path name. Possible "Actions" are read, write, and execute.
- Socket
- Window
- Property
- Runtime
- Security
- All
- Aglet
- Message
- Context
- Protection (Aglet)
- Protection (Message)

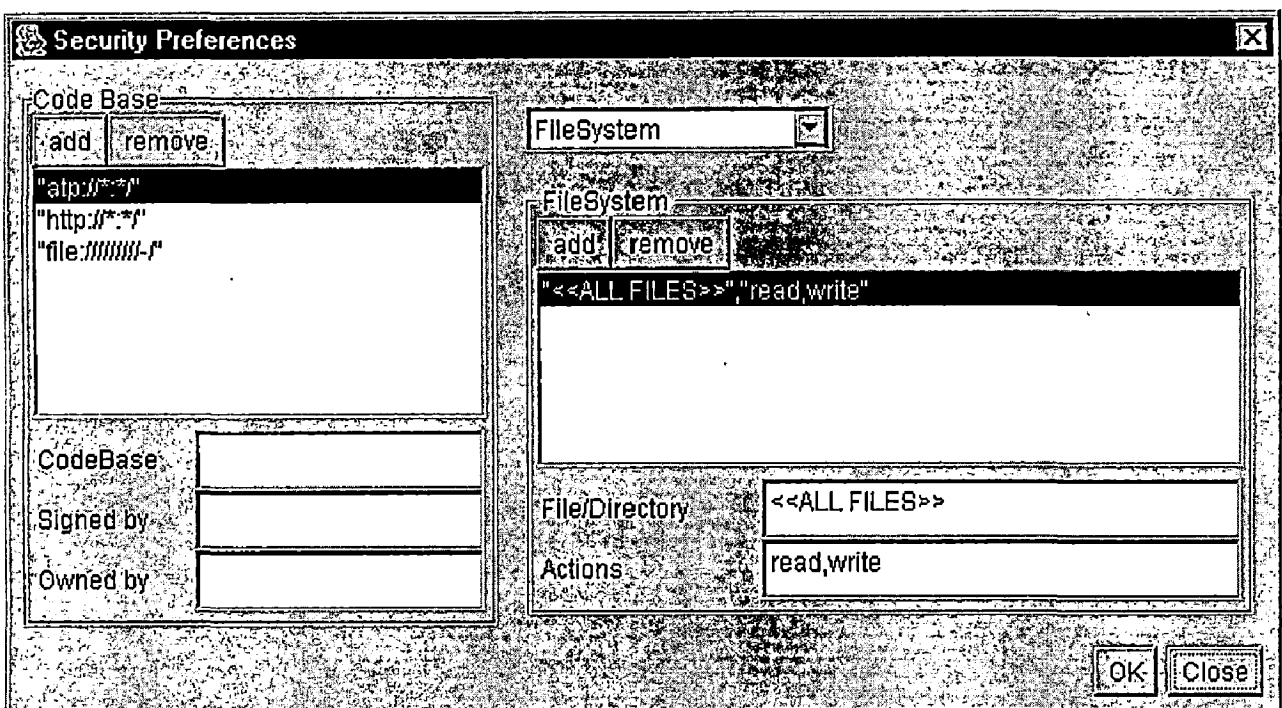


Fig.A.7.3: Security Preferences

A.7.4 Server Preferences

User can assign a public path name to a directory as an alias.

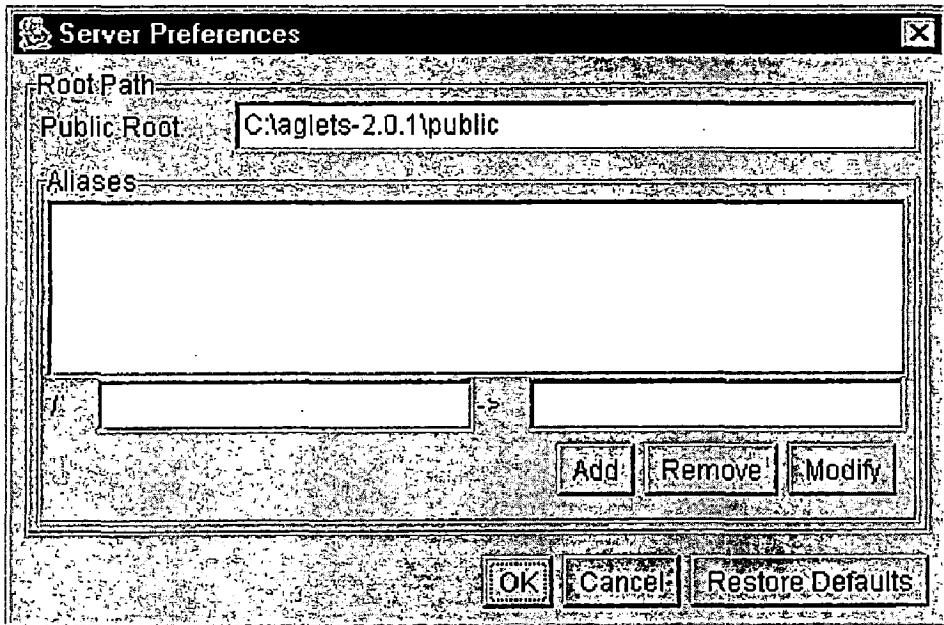


Fig.A.7.4: Server Preferences